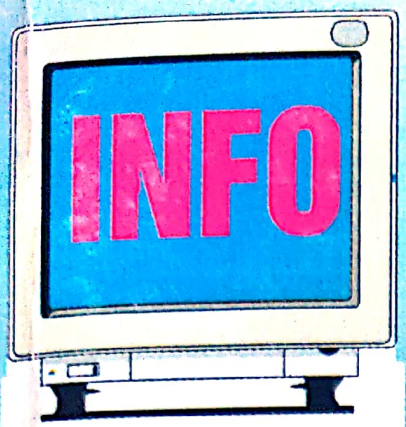


ADRIAN ATANASIU
RODICA PINTEA

CULEGERE DE PROBLEME PASCAL

PETRION

Scanned with OKEN Scanner

Iavorschi Angela

ADRIAN ATANASIU RODICA PINTEA

CULEGERE DE PROBLEME PASCAL

Profil informatică

EDITURA PETRION
Bucureşti
C.P. 1–310.Cod 70700
Tel.: 637.23.34.
Fax: 312.45.25

Scanned with OKEN Scanner

Coordonatorul colecţiei INFO conf. dr. Adrian ATANĂSIU

Referenţi: Vasile BUTNARU
Iuliu VASILESCU
Virgil PALANCIUC

Toate drepturile rezervate Editurii PETRION

ISBN 973-9116-05-1

Scanned with OKEN Scanner

CUVÂNT ÎNAINTE

Prin conţinutul prezentei cărţi, autorii urmăresc etapele învăţării limbajului de programare PASCAL. Pentru a mări funcţionalitatea lucrării, întregul material este ordonat pe capitole şi, în cadrul fiecărui capitol, pe grade de dificultate.

Fiecare problemă este însoţită de un cod în care se precizează nivelul minim de cunoştinţe necesar abordării ei (indicat prin clasă) şi eventual simbolul **C** pentru problemele de concurs.

Problemele sunt urmate, după caz, de soluţii, comentarii sau indicaţii. În special la începutul fiecărui capitol, autorii s-au străduit să prezinte cât mai explicit metodologia realizării programelor, în cadrul capitolelor fiind abordate stiluri de programare diferite.

Aducem alese mulţumiri celor trei referenţi, membrii ai lotului naţional de informatică din anul 1995, care au fost alături de noi la realizarea unor capitole. Astfel, au elaborat comentarii şi rezolvări: Vasile Butnaru – la capitolul de programare dinamică, Iuliu Vasilescu – la capitolul de recursivitate şi Virgil Palanciuc – la capitolul de grafuri.

Adresându-se cu precădere elevilor cursului liceal, ne exprimăm speranţa că lucrarea va fi utilă şi colegilor noştri ca material auxiliar la clasă.

De altfel, cartea poate fi folosită în egală măsură de studenţii aflaţi în etapa învăţării unui prim limbaj de programare, de elevii pasionaţi din cursul gimnazial şi de oricine doreşte să stăpânească tehnicile de programare.

Lucrarea fiind perfectibilă, orice sugestie utilă – trimisă pe adresa Editurii PETRION – va fi primită cu interes.

Bucureşti, 15 august 1996

AUTORII

Scanned with OKEN Scanner

I. ALGORITMI ELEMENTARI

Probleme rezolvate

1.1 (V)

Un melc se plimbă printr-un parc pe vreme umedă, viteza sa, exprimată în Km/săptămână, fiind cunoscută. Cunoscând şi lungimea totală a aleilor parcului exprimată în m, să se calculeze de câte ore are nevoie melcul pentru a străbate întregul parc.

R 1.1

Rezolvarea matematică pune în evidenţă următoarele etape:

- transformarea distanţei din m în Km;
- timpul = distanţa(Km) / viteză (Km/săptămână); se obţine durata exprimată în săptămâni;
- transformarea timpului din săptămâni în ore.

Realizarea unui astfel de program presupune cunoaşterea structurii de principiu a unui program, a instrucţiunilor de citire/scriere şi de atribuire, respectând regulile de formare a expresiilor aritmetice specifice limbajului de programare folosit.

```
program melc;
var v,l,t:real;
begin
     write('Care este viteza melcului - in km/saptamana - ?');
     readln(v);
     write('Care este lungimea parcului - in m - ?');
     readln(l);
     l:= l/1000;
     t:= l/v;
     t:= t*7*24;
     writeln('Nr. de ore necesar este:' , t:10:3)
end.
```

Observaţie: Ne-am permis momentan să nu luăm în discuţie situaţia în care, de la dispozitivul de intrare se furnizează pentru viteză valoarea 0 (când execuţia programului eşuează datorită împărţirii la zero), sau o valoare negativă pentru viteză sau lungime, caz în care rezultatul nu reflectă un caz posibil din punct de vedere experimental. Ramificarea algoritmului cu ajutorul instrucţiunii de decizie (IF) ar fi permis încheierea execuţiei cu un mesaj explicativ în oricare dintre situaţiile mai sus menţionate.

1.2 (VI)

Să se calculeze suma a două unghiuri având mărimile (exprimate în grade, minute şi secunde), cunoscute.

R 1.2

Rezolvarea presupune calculul sumelor reprezentând gradele, minutele şi secundele şi apoi ajustarea sumelor de secunde sau minute ce depăşesc valoarea

Scanned with OKEN Scanner

60. Prima dintre rezolvări presupune datele de intrare corecte (ajustate) şi se recomandă acelora ce nu cunosc încă noţiunea de „parte întreagă" (nu este exclus să o înţelegeţi urmărind – trasând – execuţia celui de-al doilea program pentru date numerice cum ar fi g1 = 12, m1 = 100, s1 = 40, g2 = 45, m2 = 25, s2 = 145:

Rezolvare 1:

- Citeşte g1,m1,s1
- Citeşte g2,m2,s2
- g=g1+g2
- m=m1+m2
- s=s1+s2
- dacă s>=60
  atunci s=s-60
  m=m+1
  sfârsit_dacă
- dacă m>=60
  atunci m=m-60
  g=g+1
  sfârsit_dacă
- scrie g,m,s

Rezolvare 2:

- Citeşte g1,m2,s1
- Citeşte g2,m2,s2
- g=g1+g2
- m=m1+m2
- s=s1+s2
- m=m+[s/60]
- s=s mod 60 (restul împărţirii la 60)
- g=g+[m/60]
- m=m mod 60
- scrie g,m,s

Observaţie: [s/60] reprezintă partea întreagă a rezultatului expresiei s/60.

```
program unghiuri_1;
var g1,g2,g,,m1,m2,m,s1,s2,s:integer;
begin
     write('Marimea unghiului I (grade minute secunde <CR>):');
     readln(g1,m1,s1);
     write('Marimea unghiului II (grade minute secunde <CR> ):');
     readln(g2,m2,s2);
     g:=g1+g2; m:=m1+m2; s:=s1+s2;
     if s>60 then begin m:=m+1;
                  s:=s-60 end;
     if m>60 then begin g:=g+1;
                  m:=m-60 end;
     writeln ('Suma : ',g,' grade ',m, ' minute', s, ' secunde')
end.
```

```
Program unghiuri_2;
var g1,g2,g,,m1,m2,m,s1,s2,s:integer;
begin
     write('Marimea unghiului I (grade minute secunde <CR>):');
     readln(g1,m1,s1);
     write('Marimea unghiului II (grade minute secunde <CR> ):');
     readln(g2,m2,s2);
     g:=g1+g2; m:=m1+m2; s:=s1+s2;
     m:= m+s div 60; s:= s mod 60;
     g:= g+ m div 60; m:= m mod 60;
     writeln('Suma :', g, ' grade ', m ,' minute ' , s , ' secunde')
end.
```

1.3 (VII)

Fiind furnizate trei numere întregi, să se stabilească dacă acestea pot fi măsurile laturilor unui triunghi şi, în caz afirmativ, să se calculeze fie perimetrul, fie aria, fie să se determine natura triunghiului după laturi (isoscel, echilateral, oarecare) în funcţie de dorinţa utilizatorului.

Scanned with OKEN Scanner

R 1.3

Condiţiile ca numerele să reprezinte mărimile laturilor unui triunghi sunt:
a) fiecare număr este strict pozitiv şi
b) fiecare mărime de latură este mai mare decât suma celorlaltor două.

Condiţia ca fiecare latură să fie mai mică decât modulul diferenţei celorlaltor două este o consecinţă a condiţiei (b). Aria se calculează cu formula lui Heron: $\sqrt{p(p-a)(p-b)(p-c)}$, unde p este semiperimetrul triunghiului.

```
program triunghi;
uses crt;
var a,b,c:integer;
    p,aria:real;
    ch:char;
begin
     write('Introduceti cele 3 nr. întregi:');
     readln(a,b,c);
     if (a>0)and(b>0)and(c>0)and(a<b+c)and(b<a+c)and(c<a+b) then
        begin
           writeln('Nr. pot fi marimile laturilor unui triunghi ');
           writeln('1. Perimetrul triunghiului');
           writeln('2. Aria triunghiului');
           writeln('3. Natura triunghiului');
           writeln('Apasati tasta-cifra corespunzatoare optiunii:');
           ch:=readkey;
           case ch of
                '1': writeln('perimetrul triunghiului este ',a+b+c);
                '2': begin
                          p:=(a+b+c)/2;
                          aria:=sqrt(p*(p-a)*(p-b)*(p-c));
                          writeln('Aria triunghiului= ',aria:15:2)
                     end;
                '3': if (a=b) and (a=c) then
                        writeln('triunghi echilateral')
                     else if (a=b) or (a=c) or (b=c) then
                        writeln('triunghi isoscel')
                     else writeln('triunghi oarecare')
           end
        end
     else writeln('Nr. nu pot fi marimile laturilor unui triunghi')
end.
```

1.4 (IX)

Cunoscând mărimea în octeţi (bytes) a unui joc aflat pe hard-discul său, un elev doreşte să citească din el segmente de lungime (exprimată tot în octeţi) cât mai mari, dar nu mai mari de m octeţi, astfel încât, citind un număr întreg de astfel de segmente, să nu-i mai rămână octeţi necitiţi. Ajutaţi-l să determine lungimea unui astfel de segment şi numărul de segmente citite.

R 1.4

Algoritmul determină cel mai mare divizor al unui număr dat l, dintr-un interval dat de numere naturale (1, .., m).

Scanned with OKEN Scanner

```
program segmente;
var l,m,i,max:longint;
begin
     write('Lungimea fisierlui exprimata in octeti:');
     readln(l);
     write('Lungimea maxima a segmentului exprimata in octeti:');
     readln(m);
     for i:=m downto 1 do
         if l mod i=0 then
            begin
                 max:=i;
                 i:=1       {sau break in Turbo Pascal 7.0}
            end;
     writeln(' Segmentul cautat are ',max,' octeti');
     writeln(' Exista ',l div max,' astfel de segmente')
end.
```

1.5 (VII)

Se citesc de la tastatură două numere reale şi apoi un caracter reprezentând una dintre operaţiile + (adunare), - (scădere), * (înmulţire), / (împărţire). Calculatorul va afişa operaţia efectuată cu cele două numere date, precum şi rezultatul ei.

R 1.5

Un algoritm care presupune luarea unei decizii din mai multe posibile este:

```
program operatii;
var a,b:real;z:byte;
    o:char;
begin
     write('a='); readln(a);
     write('b='); readln(b);
     write('Operatia (+ - * /) :'); readln(o);
     write('Cu cate zecimale doriti sa lucrati? '); readln(z);
     write(a:z+6:z,o);        {cu z zecimale pe z+6 pozitii}
     if b<0 then write('(',b:z+6:z,')=')
            else write(b:z+6:z,'=');
     case o of
          '+': writeln (a+b:z+6:z);
          '-': writeln (a-b:z+6:z);
          '*': writeln (a*b:z+6:z);
          '/': if b=0 then writeln('eroare')
                      else writeln(a/b:z+6:z)
          else writeln('operatie eronata ')
     end
end.
```

1.6 (VIII)

Rezolvarea ecuaţiilor de forma $ax + b = 0$ (ecuaţii de gradul I) reprezintă un lucru banal în matematică dacă se cunosc coeficienţii a şi b. Cum procedează calculatorul ? Scrieţi programul corespunzător.

R 1.6

Algoritmul trebuie gândit identic situaţiei rezolvării unei ecuaţii de gradul întâi cu parametrii a şi b:

Scanned with OKEN Scanner

```
program  ec_gr_1;
var a,b,x:real;
begin
     write('a='); readln(a);
     write('b='); readln(b);
     if a<>0 then
        begin x:=-b/a; writeln('x=',x:10:3) end
     else if b=0  then writeln('exista o infinitate de solutii')
          else writeln('nu exista solutii');
     readln
end.
```

1.7 (IX, Tudor Sorin – Manual clasa a IX-a)

Să se calculeze valoarea funcţiei f definită pe mulţimea numerelor reale, pentru un x citit:

$$f = \begin{cases} x^2 - 3, \text{ pentru } x < 5 \\ x + 1, \text{ pentru } 5 \le x \le 25 \\ x - 5x + 6, \text{ pentru } x > 25 \end{cases}.$$

R 1.7

Programul foloseşte structuri **IF** imbricate (şi este un prilej de a gândi eficient pe principiul că, dacă am pus întrebarea „Moneda a căzut cu stema în sus ?" şi ni s-a răspuns NU, atunci nu este cazul să mai întrebăm dacă a căzut cu cealaltă faţă în sus):

```
program functie;
var x,f:real;
begin
     write ('dati argumentul functiei:');
     readln(x);
     if x<5 then f:=sqr(x)-3
     else if x<=25 then f:=x+1
          else f:=x*x-5*x+6;
     writeln ('valoarea functiei este :',f:7:2);
     readln
end.
```

1.8 (V – C, Concurs „Grigore Moisil", Lugoj, 1995)

A fost odată un ţăran sărac. Într-o zi a găsit o sămânţă. A pus-o în pământ, şi minune: după ce a răsărit a început să crească în înălţime, în fiecare zi câte un metru! După două zile apare un mugur cu o nouă ramură şi apoi în fiecare zi mai apare câte o ramură. Fiecare ramură nouă creşte şi înmugureşte la fel ca prima. După câteva zile ţăranul taie planta şi are dificultăţi în a număra ramurile şi a măsura lungimea lor totală.

Ţăranul vă roagă să-l ajutaţi: el vă spune numărul de zile după care a tăiat planta şi voi să îi spuneţi câte ramuri are şi ce lungime totală obţine dacă ar pune cap la cap toate ramurile ?

Exemplu: după 5 zile planta are 5 ramuri totalizând 12 m.

R 1.8

Calculând numărul de ramuri pentru primele patru-cinci zile se observă că acesta se obţine prin adunarea numărului de ramuri existente în ziua precedentă cu

Scanned with OKEN Scanner

cele noi (apărute din fiecare ramură existentă în urmă cu două zile). De unde formula *nr_curent=nr_precedent+nr_anteprecedent*. Lungimea totală va fi lungimea din ziua precedentă la care se adaugă câte un metru pentru fiecare ramură din ziua curentă.

Observaţie: numărul de ramuri existente în fiecare zi formează un şir de numere cu proprietăţi interesante, care poartă numele matematicianului Fibonacci.

```
program ramuri;
var n,i:byte;
    nr_curent,nr_precedent,nr_anteprecedent,lung:longint;
begin
     write('numar zile=');readln(n);
     nr_curent:=0;nr_precedent:=1;lung:=0;
     for i:=1 to n do
         begin
              nr_anteprecedent:=nr_precedent;
              nr_precedent:=nr_curent;
              nr_curent:=nr_anteprecedent+nr_precedent;
              lung:=lung+nr_curent
         end;
     writeln('numar de ramuri=',nr_curent);
     writeln('lungime totala=',lung)
end.
```

1.9 (V)
Să se afişeze suma cifrelor unui număr natural dat.

R 1.9
Se detaşează fiecare cifră de la sfârşitul numărului calculând restul împărţirii numărului la 10; numărul rămas după desprinderea cifrei este câtul împărţirii la 10:

```
program suma_cifrelor;
var n,s:integer;
begin
     write('n=');
     readln(n);
     s:=0;
     while n<>0 do
     begin
          s:=s+n mod 10;
          n:=n div 10;
     end;
     writeln('s=',s);
     readln
end.
```

1.10 (V)
Să se determine toţi divizorii primi ai unui număr natural dat.

R 1.10
Algoritmul prin care căutăm toţi divizorii verificând pentru fiecare dacă e prim este neeficient. Este preferabil algoritmul prin care se împarte numărul cu fiecare divizor găsit, astfel nemaifiind necesar să testăm dacă aceşti divizori sunt primi.

```
program divizori_primi;
var n,i: word;
```


Scanned with OKEN Scanner

```
begin
     write('Dati un numar natural: '); readln(n);
     i:=2;
     while n>1 do begin
           if n mod i = 0 then begin
              repeat
                    n:=n div i
              until n mod i<>0;
              writeln(i)
           end;
           i:=i+1
     end
end.
```

1.11 (V)

Să se scrie descompunerea în factori primi *a* unui număr natural dat.

R 1.11

În programul anterior se declară variabila e (iniţializată cu 0 în corpul instrucţiunii *if n mod i* = 0...) şi se incrementează (e:=e+1) în corpul instrucţiunii repeat... Se va folosi o scriere *write*(i,'^',e,'+'), iar în final se adaugă o scriere *writeln*(#8,' ') care merge un caracter înapoi şi tipăreşte un spaţiu pentru a anula ultimul '+' scris inutil.

1.12 (VI)

Se citeşte un număr natural de cel mult 9 cifre şi se afişează o piramidă formată din cifrele acestui număr: pe prima linie cifra (sau cifrele) din mijloc, pe a doua linie cele 3 (sau 4) cifre din mijloc etc., pe ultima linie fiind scris întregul număr. Se va asigura aspectul de triunghi isoscel plin al figurii.

Exemplu: dacă este dat numărul 237855, se afişează:

```
  78
 3785
237855
```

R 1.12

În rezolvarea problemei se poate porni de la ideea de a detaşa din număr cifrele corespunzătoare de pe extreme (prin împărţiri şi înmulţiri cu puterile corespunzătoare ale lui 10), afişarea acestora făcându-se apoi cu un format de scriere variabil:

```
program piramida1;
var e,i,j,c:byte;    {c=nr de cifre} {r numarul dupa eliminare}
    n,nn,f,r:longint;{e=nr.maxim de cifre eliminate pe o latura}
begin
     write('dati un numar de cel mult 9 cifre:'); readln(n);
     c:=0; nn:=n;
     repeat
           c:=c+1; nn:=nn div 10
     until nn=0;
     e:=(c-1) div 2;
     for i:=e downto 0 do
         begin
```


Scanned with OKEN Scanner

```
            f:=1;for j:=1 to I do f:=f*10;
            r:=n div f;    {eliminarea cifrelor de la sfarsit};
            f:=1;
            for j:=1 to c-2*i do f:=f*10;
            r:=r mod f;    {eliminarea primelor cifre}
            writeln(r:c-i)
        end
end.
```

Acelaşi efect, dar cu un suport matematic modest, poate fi obţinut prin ştergerea cifrelor chiar de pe ecran (amintim că efectul caracterului cu codul 8 -#8- este acela de a muta cursorul de scriere cu o poziţie înapoi, iar al caracterului cu codul 13 -#13- este acela de a trece la capătul rândului curent, fără a trece deci pe rândul următor, trecerea pe rând nou realizându-se la tipărirea caracterului #10):

```
var m,j,k,c:byte;          {c=numarul de cifre; m=pozitia din mijloc}
    n,nn:longint;
begin
     write('dati un numar de cel mult 9 cifre:'); readln(n);
     c:=0; nn:=n;
     repeat
           c:=c+1; nn:=nn div 10
     until nn=0;
     m:=(c+1) div 2;
     for j:=1 to m do
         begin
              write(n);
              for k:=1 to m-j do write(#8);
              for k:=1 to m-j do write(' ');
              write(#13);
              for k:=1 to m-j do write(' ');
              writeln
         end
end.
```

1.13 (VI)

Fiind date un număr natural şi un număr prim, să se stabilească la ce putere apare factorul prim în descompunerea numărului dat.

Observaţie: Un factor care nu apare în descompunerea unui număr va fi considerat la puterea 0.

R 1.13

Algoritmul de testare a proprietăţii de număr prim pentru un număr natural *n* poate fi conceput şi ceva mai simplu, fără testarea paritătii numărului; în acest caz însă, criteriul de căutare al divizorului până la radical din acel număr devine obligatoriu pentru ca programul să poată rula în timp „suportabil" pentru valori mari ale lui *n*. Atragem atenţia asupra numărului 1 care nu este prim.

```
program multiplicitate;
var n,f,m:integer;
    prim:boolean;
begin
     repeat
          write('introduceti un numar natural:');
          readln(n);
```


Scanned with OKEN Scanner

```
      until n>0;
      repeat
            write ('introduceti un factor prim:');readln(f);
            if f=1 then prim:=false else prim:=true;
            if odd(f) then
               begin i:=3; pas:=2 end
            else
                begin i:=2; pas:=1 end;
            while prim and (i<=sqrt(f)) do
                  if f mod i =0 then prim:=false
                                else i:=i+pas

      until prim;
      m:=0;
      while n mod f=0 do
            begin n:=n div f; m:=m+1 end;
      writeln('numarul ',f,' apare la puterea ',m);
      readln
end.
```

1.14 (VI – C)

Să se scrie toate numerele n până la 1000 al căror pătrat se termină cu n.

R 1.14

Algoritmul determină dacă numărul $n^2 - n$ se termină cu un număr de zerouri egal cu numărul de cifre al lui n.

```
program patrat;
var n:integer;
begin
      for n:=1 to 9 do
          if n*(n-1) mod 10=0 then writeln(n);
      for n:=11 to 99 do
          if n*(n-1) mod 100=0 then writeln(n);
      for n:=101 to 999 do
          if n*(n-1) mod 1000=0 then writeln(n)
end.
```

1.15 (VII – C)

Fiind dată valoarea unui număr natural, se cere să se tipărească această valoare în scriere romană.

R 1.15

În algoritm se urmăreşte scrierea şi detaşarea ordinelor romane de la cifrele cele mai mari până la cele mai mici (M=1000, C=100, X=10, I=1), evidenţiind separat cifrele D=500, L=50, V=5 şi grupurile CM=900, XC=90, IX=9 care pot să apară o singură dată în reprezentarea romană a numărului:

```
program nr_roman;
var  x:word;
begin
      write ('Dati numarul arab:');
      readln (x);
      write(x,'=');
      while x>=1000 do begin write('M'); x:=x-1000 end;
```


Scanned with OKEN Scanner

```
        if x>=900 then begin write('CM'); x:=x-900 end
        else
            if x>=500 then begin write('D'); x:=x-500 end
            else
                if x>=400 then begin write('CD'); x:=x-400 end;
        while x>=100 do begin write('C'); x:=x-100 end;
        if x>=90 then begin write('XC'); x:=x-90 end
        else
            if x>=50 then begin write('L'); x:=x-50 end
            else
                if x>=40 then begin write('XL'); x:=x-40 end;
        while x>=10 do begin write('X'); x:=x-10 end;
        if x>=9 then begin write('IX'); x:=x-9 end
        else
            if x>=5 then begin write('V'); x:=x-5 end
            else
                if x=4 then begin write('IV'); x:=x-4 end;
        while x>=1 do begin write('I'); x:=x-1 end;
        readln
end.
```

1.16 (V – C)

Se citesc mai multe caractere până la întâlnirea caracterului ')'. Verificaţi dacă expresia formată din înlănţuirea acestor caractere reprezintă un număr zecimal periodic. În caz afirmativ să se determine două numere întregi prime între ele al căror raport este chiar numărul zecimal dat. Se vor folosi codurile ASCII ale caracterelor '(' -> 40, ')' -> 41, '.' -> 46, '0'..'9' -> 48..57.

Exemplu:

```
2.3(1)
```

Este o expresie corectă:

```
104 si 45
```

R 1.16

Se reţin prin citirea caracter cu caracter 3 numere importante: numărul reprezentând partea întreagă (nr), numărul reprezentând partea zecimală neperiodică (zec) şi numărul reprezentând partea zecimală periodică (per). Formula matematică de transformare a unui număr zecimal în fracţie este:

$$a_1 \dots a_k, z_1 \dots z_n\,(p_1 \dots p_l) = a_1 \dots a_k + \frac{z_1 \dots z_n p_1 \dots p_l - z_1 \dots z_n}{\underbrace{9 \dots 9}_{l \text{ ori}}\,\underbrace{0 \dots 0}_{n \text{ ori}}}$$

```
program expresie;
var corect :boolean;
    i,z,p:char;
    n,l,j:byte;
    nr,zec,per,f,nmr,nmt,deimp,imp,rest:integer;
begin
     writeln('dati un nr zecimal');
     nr:=0;zec:=0;per:=0;n:=0;l:=0;
     corect:=true;
```


Scanned with OKEN Scanner

```
repeat
      read(i);
      case I of
           '0'..'9' : Nr:=nr*10+ord(i)-48;
           '.'      :       {instructiune vida}
           else corect:=false
      end
until (i='.') Or (i=')');
if not corect then
   begin writeln('nu este corecta'); halt end;
repeat
      read(z);
      case z of
           '0'..'9':
              begin zec:=zec*10+ord(z)-48; n:=n+1 end;
           '(':
           else corect:=false
       end
until (z='(') or (z=')');
if not corect then
   begin writeln('nu este corecta'); halt end;
repeat
      read(p);
      case p of
           '0'..'9':
             begin per:=per*10+ord(p)-48; l:=l+1; end;
           ')':
           else corect:=false
      end;
until p=')';
if not corect then
   begin writeln('nu este corecta'); halt end;
writeln('e corecta');  {nmt=numitorul, nmr=numaratorul}
f:=1;
for j:=1 to l do f:=f*10;
nmt:=zec* f+per; nmt:=nmt-zec; nmr:=0;
for j:=1 to l do nmr:=nmr*10+9;
for j:=1 to n do nmr:=nmr*10;
nmt:=nmt + nr * nmr;     {introducerea intregilor in fractie}
deimp:=nmt; imp:=nmr;    {algoritmul lui Euclid}
rest:=deimp mod imp;
while rest<>0 do
  begin deimp:=imp;
        imp:=rest;
        rest:=deimp mod imp end;
nmt:=nmt div imp;                          {simplificarea}
nmr:=nmr div imp;
writeln(nmt,' si ',nmr)
end.
```

Observaţie: în rezolvare nu s-a considerat cunoscut tipul de date string. Dacă şirul de caractere introdus este foarte mare, nu se mai poate lucra numeric cu cele 3 secţiuni, ci se impun: memorarea lor sub forma de stringuri de cifre, simularea scăderii pe astfel de structuri şi modelarea algoritmului lui EUCLID la o secvenţă de scăderi succesive.

Scanned with OKEN Scanner

1.17 (IX – C, Olimpiada la clasa a IX-a, faza pe sector, Bucureşti, 1993)
Se introduce de la tastatură un număr natural. Să se determine numărul minim şi numărul maxim care se obţin prin eliminarea unei singure cifre din numărul dat.

R 1.17

Algoritmul sugerat de enunţul problemei constă în memorarea în *nmax* şi *nmin* a valorii maxime, respectiv minime din şirul numerelor obţinute prin eliminarea succesivă a câte unei cifre din numărul dat:

```
program elimin_cifra1;
var n,nn,f,nmin,nmax,nr_redus:longint;
    nr_cif,i,j:byte;
    cifra_curenta,cifra_precedenta:0..9;
begin
     repeat
          write('introduceti un numar natural cu 2..8 Cifre:');
          readln(n);
     until n>9;
     nr_cif:=0; nn:=n;  {calculeaza nr.De cifre ale lui n}
     repeat
          nr_cif:=nr_cif+1; nn:=nn div 10
     until nn=0;
     nmin:=maxlongint;nmax:=0;
     for i:=0 to nr_cif-1 do  {eliminarea cifrei de rang i}
         begin
              f:=1;
              for j:=1 to I do f:=f*10;
              if n div (f*10)*f+n mod f>nmax then
                 nmax:=n div (f*10)*f+n mod f;
              if n div (f*10)*f+n mod f<nmin then
                 nmin:=n div (f*10)*f+n mod f
         end;
     writeln('numarul maxim este:',nmax);
     writeln('numarul minim este:',nmin);
     readln
end.
```

Dacă studiem relaţiile existente între cifra eliminată şi celelalte cifre, vom concepe un algoritm mai eficient, deşi mai laborios. Eliminăm pentru minim prima cifră care este urmată de o cifră cu valoare mai mică, iar pentru maxim pe cea urmată de o cifră mai mare. În cazul în care nu există o astfel de cifră, se elimină ultima cifră. Am preferat studiul perechilor de cifre de la dreapta la stânga pentru a simplifica formulele de calcul, în această situaţie memorând în vederea eliminării rangul cifrei precedente în studiul perechii *cifră_curentă-cifră_precedentă* (pentru alte detalii, precum şi generalizarea acestui algoritm, a se vedea V. Mitrana, „Provocarea algoritmilor", Ed. Agni, 1993):

```
program elimin_cifra2;
var n,nn,f:longint;
    nr_cif,pmin,pmax,i:byte;
    cifra_curenta,cifra_precedenta:0..9;
begin
     repeat
          write('introduceti un nr. format din mai multe cifre:');
          readln(n)
     until n>9;
```


Scanned with OKEN Scanner

```
    nr_cif:=0; nn:=n;  {calculeaza nr. de cifre ale lui n}
    repeat
          nr_cif:=nr_cif+1; nn:=nn div 10
    until nn=0;
    pmin:=0;pmax:=0;i:=0; {pozitia (rangul) ultimei cifre}
    nn:=n;
    cifra_curenta:=n mod 10;
    cifra_precedenta:=n mod 100 div 10;
    while nn>9 do begin
          if cifra_curenta<cifra_precedenta then pmin:=i+1
                                            else pmax:=i+1;
          nn:=nn div 10;
          cifra_curenta:=cifra_precedenta;
          cifra_precedenta:=nn mod 100 div 10;
          i:=i+1
    end;
    f:=1;for i:=1 to pmax do f:=f*10;
    writeln('numarul maxim este:',n div (f*10)*f+n mod f);
    f:=1;for i:=1 to pmin do f:=f*10;
    writeln('numarul minim este:',n div (f*10)*f+n mod f)
end.
```

Celor care cunosc deja lucrul cu şiruri le lăsăm plăcerea de a rescrie programul mult simplificat şi aplicabil la numere oricât de mari.

1.18 (IX – C, Olimpiada la clasa a IX-a, faza pe sector, Bucureşti, 1995)

Se consideră două numere întregi. Să se stabilească dacă acestea sunt termeni consecutivi în şirul Fibonacci (Şirul care are primii doi termeni egali cu 1, fiecare dintre următorii termeni fiind suma dintre termenul precedent şi termenul anteprecedent din şir).

R 1.18

O primă idee conduce la algoritmul care construieşte elementele şirului până la termenul care este mai mare sau egal cu cel mai mic dintre numerele date; în cazul egalităţii se calculează şi termenul următor al şirului şi dacă acesta este egal cu cel mai mare dintre numerele date, răspunsul este 'DA'; în toate celelalte cazuri răspunsul este 'NU'.

O altă idee pleacă de la numerele date, calculează elementele şirului „înapoi" după formula *termen=diferenţa-termenilor-imediat-următori* şi se opreşte când se ajunge la doi termeni succesivi egali cu 1 ('DA') sau un termen mai mic decât 1 ('NU').

```
program Fibonacci;
var a,b,c:longint;
begin
     write('Dati numerele:');readln(a,b);
     if a>b then
        begin                          {ORDONAREA NUMERELOR}
             c:=a; a:=b; b:=c end;
     while b>1 do
           begin c:=b-a; b:=a; a:=c end;
     if (b=1)and(a=1) then writeln('DA')
                      else writeln('NU')
end.
```


Scanned with OKEN Scanner

1.19 (VIII – C)

Se calculează „cifra de control" a unui număr întreg efectuând suma cifrelor sale, apoi suma cifrelor acestei sume etc. până când se obţine o sumă formată dintr-o singură cifră. Să se determine câte dintre numerele de n cifre (n natural dat) au cifra de control egală cu k (k cifră dată).

R 1.19

Algoritmul în care se calculează cifra de control pentru fiecare număr de n cifre, numărând pe cele care au această cifră egală cu k poate fi considerat „falimentar" din punctul de vedere al timpului de execuţie.

Analiza problemei conduce la o soluţie matematică: toate cifrele de control pentru numere succesive parcurg circular mulţimea {1,2,..,9}. Numerele de n cifre sunt cele aflate între 10^{n-1} şi $10^n - 1$, cifra de control a numărului 10^{n-1} este 1, a lui $10^n - 1$ este 9 şi există $10^n - 10^{n-1} = 10^{n-1} * 9$ astfel de numere; dintre acestea, $10^{n-1} * 9 / 10$ vor avea cifra de control egală cu k (indiferent cât ar fi acest k).

Deci răspunsul este $10^{n-2} * 9$, adică se tipăreşte un 9 urmat de $n - 2$ cifre de 0. Cazul $n = 1$ se analizează separat. Algoritmul funcţionează pentru n oricât de mare.

```
program cifra_de_control;
var n,i:longint;
begin
     write('numarul de cifre:');
     readln(n);
     if n=1 then writeln(1)
     else begin
               write(9);
               for i:=1 to n-2 do write(0);
               writeln
          end
end.
```

1.20 (IX)

(Pentru că prea multe lucruri serioase devin obositoare!)

Să se genereze cu ajutorul calculatorului un număr natural mai mic decât 100. Să se simuleze un joc de ghicire a acestui număr, operatorul introducând un număr de la tastatură, iar calculatorul răspunzându-i printr-un sunet a cărui frecvenţă reprezintă numărul generat şi un sunet a cărui frecvenţă reprezintă numărul dat. Sunetele trebuie să se poată distinge pentru două numere consecutive şi formula de calcul a tonalităţii relative a fiecărui sunet trebuie să încadreze toate sunetele în gama auditivă. Jocul se încheie când a fost „ghicit" numărul generat de către calculator. Se vor afişa: numărul total de încercări şi numărul de încercări „afonice" (aflate în celălalt interval decât cel logic dintre cele două în care se împarte intervalul [0,100] ca urmare a numărului dat de operator).

Scanned with OKEN Scanner

R 1.20

Rezolvarea problemei presupune cunoaşterea procedurii randomize şi a funcţiei random, lucrul cu unitatea crt pentru apelarea procedurilor *sound*, *delay* şi (mai ales!) *nosound*. Pentru a stabili dacă o nouă încercare este „fonică" sau nu, se memorează în variabila *xx* încercarea precedentă:

```
program ghici;
uses crt;
var xx:integer;n,x,c,f:byte;
begin
     randomize;
     c:=0;f:=0; n:=random(100); xx:=-1;
     repeat
          write('dati numarul:'); readln(x);
          writeln('numarul corect este:');
          sound(30+n*30);delay(500);nosound;delay(300);
          writeln('numarul tau este:');
          sound(30+x*30);delay(500);nosound;
          if (x>xx) xor (xx<n) then f:=f+1;
          c:=c+1;xx:=x
     until n=x;
     writeln('ai ghicit din ',c,'incercari!');
     If f=0 then writeln('ai un auz impecabil!')
         else writeln('de ',f,' ori te-a inselat auzul');
     readln
end.
```

Probleme propuse

1.21 (VI)

Se citeşte un număr reprezentând capacitatea (în cm^3) unui recipient şi apoi se citesc pe rând, densităţile (în Kg/m^3) şi cantităţile (în mg) unor substanţe chimice. Să se determine a câta dintre substanţele date încape în recipient astfel încât acesta să fie cât mai plin.

1.22 (VII)

O navă parcurge distanţa d (în Km) dintre două staţii orbitale în a săptămâni şi b zile. Să se determine viteza navei exprimată în m/s.

1.23 (IX)

Un corp cu volumul *v* este format din *n* ($n <= 100$) materiale diferite. Dacă se citesc de la tastatură numerele *n* şi *v* şi apoi n perechi reprezentând densitatea şi masa fiecărui material care intră în componenţa sa, să se afişeze pentru fiecare pereche procentul din volumul corpului reprezentat de volumul materialului respectiv şi în final să se tipărească masa corpului.

1.24 (IX)

Se citesc trei numere. Să se stabilească dacă acestea pot fi mărimile laturilor unui triunghi şi, în caz afirmativ, să se tipărească mărimile unghiurilor acestui

Scanned with OKEN Scanner

triunghi exprimate în grade, minute şi secunde. Suma unghiurilor astfel determinate este totdeauna 180 grade?

Indicaţie: se foloseşte formula lui Pitagora generalizată: $a^2 = b^2 + c^2 - 2bc\cos(b,c)$ pentru determinarea mărimii fiecărui unghi exprimată în radiani.

1.25 (IX)

Realizaţi un program care permite introducerea repetată a două numere naturale şi a unuia dintre operatorii * (înmulţire), ^ (ridicare la putere), & (şi-logic pe reprezentarea binară a numerelor), | (sau-logic pe reprezentarea binară a numerelor), calculatorul afişând de fiecare dată operaţia corespunzătoare şi rezultatul ei.

Exemplu: Pentru numerele 15 şi 8 şi operatorul & se va afişa:

$$15 \;\&\; 8 = 8 \;(1111_2 \;\&\; 1000_2 = 1000_2)$$

1.26 (VII)

Să se scrie un program care afişează un meniu din care să se permită alegerea unei opţiuni prin care să se introducă două numere reprezentând două dintre cele 3 elemente ale unui triunghi dreptunghic (mărimea unei catete, mărimea ipotenuzei sau mărimea unui unghi în grade) şi calculatorul să determine celelalte elemente, perimetrul, aria şi înălţimea corespunzătoare ipotenuzei.

1.27 (IX)

Se citesc trei numere reprezentând laturile unui triunghi. Să se stabilească natura triunghiului după laturi şi unghiuri (ascuţitunghic, ascuţitunghic isoscel, obtuzunghic, obtuzunghic isoscel, dreptunghic isoscel, echilateral etc.).

1.28 (VI – C)

Se citesc un număr real r şi un număr natural n ($n \leq 16$). Să se afişeze numărul r cu n zecimale pe ecran în formă de piramidă (caracterul sau caracterele din mijloc pe o linie, cele două caractere de la stânga şi de la dreapta celor afişate pe linia a doua, etc., pe ultima linie afişându-se primul şi ultimul caracter ale numărului).

Exemplu:

```
r=14.63
n=4
```

Se vor afişa:

```
   6
  . 3
 4   0
1     0
```

provenind din reprezentarea numărului 14.6300.

1.29 (V – C)

Se citesc pe rând de la tastatură mai multe caractere ce pot fi numai cifre, unul dintre operatorii '+','-','*','/' sau semnul '=', citirea făcându-se până la întâlnirea semnului '='. Se consideră că şirul de caractere reprezintă o expresie corectă din punct de vedere matematic şi că operaţiile se fac în ordinea în care sunt

Scanned with OKEN Scanner

scrise (nu se ţine cont de prioritatea operatorilor de înmulţire şi împărţire). Calculatorul trebuie să afişeze rezultatul acestei expresii.

1.30 (VIII)

Se consideră un număr n scris în baza 10. Să se determine diferenţa dintre suma pătratelor cifrelor ce constituie partea sa întreagă şi suma pătratelor cifrelor zecimale.

1.31 (IX)

Dacă se cunosc coeficienţii a, b, c, a', b' şi c' ai ecuaţiilor $ax + bx + c = 0$ şi $a'x + b'x + c' = 0$, să se stabilească dacă există o rădăcină reală comună a celor două ecuaţii.

1.32 (X)

Se cunosc coeficienţii a şi b ai numărului complex $z = a + bi$. Să se determine coeficienţii numărului complex z^n, pentru n dat.

Indicaţie: Se folosesc forma trigonometrică a numerelor complexe $z = a + bi = r(\sin \alpha + i \cos \alpha)$ şi formula lui Moivre $z^n = r^n(\sin n\alpha + i \cos n\alpha)$. Se va trata separat cazul $z = 0$.

1.33 (IX)

La o societate sunt n ($n < 32$) acţionari. Cel mai important acţionar deţine din capitalul societăţii un procent concretizat prin c milioane lei ($c > 1000000$). Următorul ca importanţă dintre acţionari deţine jumătate din procentul deţinut de primul, următorul jumătate din procentul deţinut de al doilea, etc. Realizaţi cel mai scurt program cu putinţă care tipăreşte suma deţinută de ultimul acţionar.

1.34 (X)

Să se realizeze un program interactiv care permite selectarea dintr-un meniu a unui tip de progresie (aritmetică sau geometrică), apoi alegerea dintr-un submeniu a unui caz dintre cele care conduc la generarea primilor n termeni ai progresiei (dacă se dau primul şi al doilea termen, primul termen şi raţia, suma termenilor dintre doi indici daţi, etc.); să se solicite datele necesare şi numărul dorit de termeni şi în final să afişeze pe ecran aceşti termeni.

1.35 (X)

Se citesc de la dispozitivul de intrare două numere reprezentând coordonatele carteziene ale unui punct M în plan (abscisa şi ordonata). Dacă punctul se află în cadranul I sau III, se va afişa aria dreptunghiului având două vârfuri opuse în O (originea axelor) şi respectiv M; dacă punctul se află în cadranul II sau IV, se va afişa aria sectorului de cerc delimitat de axele de coordonate în cadranul respectiv (se consideră cercul având centrul în origine şi raza OM).

1.36 (XI)

Se dau coordonatele carteziene x, y şi x', y' pentru două puncte A şi B din plan. Să se determine lungimea segmentului format pe dreapta AB de punctele de intersecţie ale dreptei cu fiecare dintre axele de coordonate. Programul să prevadă şi să soluţioneze cazurile limită.

Scanned with OKEN Scanner

1.37 (V)

Să se afişeze toţi divizorii comuni pentru două numere întregi date.

1.38 (VIII)

Se consideră două numere reale m şi n. Să se determine, dacă există, cel mai mare număr prim din intervalul $[m, n]$.

1.39 (VIII)

Să se afişeze toate numerele de n cifre, care să aibă suma cifrelor n^2 şi să fie divizibile cu n^2. Problema are soluţie pentru orice n ? Deduceţi răspunsul realizând un program pentru diverse valori „experimentale" ale lui n.

Indicaţie: Numerele de n cifre se găsesc între 10^{n-1} şi $10^n - 1$. Folosind o structură FOR de la 10^{n-1} la $10^n - 1$ şi testând proprietatea cerută, programul va rula deosebit de greoi. Instrumentul matematic ne ajută să optimizăm algoritmul de căutare limitând valorile lui n la mulţimea $\{0, 1, .., 9\}$, pentru $n = 9$ deducându-se unica soluţie: 999999999.

Pentru acelaşi enunţ înlocuind n^2 cu n^3 se demonstrează că n se poate afla în mulţimea $\{0, 1, 2, 3\}$ şi există 3 valori posibile: 1, 80, 999.

1.40 (IX – C, Olimpiada la clasa a IX-a, faza judeţeană, Bucureşti, 1995)

Să se elaboreze un program care citeşte de la mediul de intrare numerele naturale m şi n ($1 \le m \le n \le 2^{31} - 1$) şi realizează următoarea prelucrare:

- Fiecărui număr (în reprezentare zecimală) cuprins între m şi n i se determină suma cifrelor, apoi suma cifrelor numărului obţinut, ş.a.m.d., până se obţine o singură cifră. Astfel, în final se va obţine un şir de numere.
- Afişează numărul elementelor distincte din şirul final precum şi numărul de apariţii ale fiecărui element din şir.

Observaţii:

1) Se va verifica corectitudinea datelor de intrare, iar algoritmul de rezolvare a problemei, descris în pseudocod, va fi inclus în textul sursă sub formă de comentariu.

2) Se cere obţinerea unui algoritm optim din punct de vedere al resurselor timp şi memorie.

Exemplu: Pentru $m = 23457$ şi $n = 23477$, şirul final conţine cifrele 1, 2, 6, 7, 8 şi 9 de două ori, iar cifrele 3, 4 şi 5 de trei ori.

Indicaţie: se folosesc observaţiile din cadrul rezolvării problemei 1.19.

1.41 (V)

Să se afişeze în centrul ecranului pe sistemul *contor cu 8 cifre*, toate numerele naturale situate între două numere întregi date, fără a folosi unitatea crt. Se vor afişa toate numerele, inclusiv cele date, în centrul ecranului, pe rând, cu o pauză oarecare între ele, rescriind pentru fiecare număr nou doar cifrele modificate.

1.42 (V-C)

Să se realizeze un efect de ecran prin tipărirea în ordine crescătoare a tuturor cifrelor folosite într-o bază b dată ($b <= 16$) astfel încât fiecare cifră să apară din marginea dreaptă a ecranului, să avanseze câte o poziţie spre stânga şi să dispară prin marginea stângă, „trăgând după ea" cifrele următoare. După ultima cifră se vor relua cifrele de la început.

Scanned with OKEN Scanner

II. ALGORITMI CU TABLOURI

Probleme rezolvate

2.1 (V)

Se citesc două şiruri de numere naturale de cel mult două cifre fiecare. Să se construiască un şir care conţine pe fiecare poziţie suma elementelor corespunzătoare din cele două şiruri date şi să se tipărească elementele acestuia în ordine crescătoare.

Exemplu: pentru şirurile 7 9 53 şi 11 2 31 se afişează şirul: 11 18 84.

R 2.1

Algoritmul de ordonare folosit poartă numele „metoda bulelor", fiind o variantă scurtă, dar mai puţin eficientă a acestuia:

```
program suma_de_vectori;
var a,b,c:array[1..100] Of byte;
    n,i,j,aux:byte;
begin
     write('Cate elemente are fiecare vector ? '); readln(n);
     writeln('Dati elem. vectorului I cu spatii sau CR intre ele:');
     for i:=1 to n do read(a[i]); readln;
     writeln('Dati elementele celuilalt vector: ');
     for i:=1 to n do read(b[i]); readln;
     for i:=1 to n do c[i]:=a[i]+b[i];
     for i:=1 to n-1 do                          {algoritm de ordonare}
         for j:=1 to n-i do
             if c[j]>c[j+1] then   {metoda celor 3 pahare}
                begin aux:=c[j]; c[j]:=c[j+1]; c[j+1]:=aux end;
     for i:=1 to n do write(c[i]:5); readln
end.
```

2.2 (V)

Să se listeze elementele care se repetă dintr-un şir numeric cu 10 elemente.

R 2.2

Un prim program este foarte simplu: se ia fiecare element al tabloului şi se compară cu toate elementele aflate după el.

```
program repetare;
var a:array[1..10] of integer;
    n,i:integer;
begin
     for n:=1 to 10 do readln(a[n]);
     for n:=1 to 10 do
         for i:=n+1 to 10 do
             if a[n]=a[i] then writeln(a[n], ' se repeta');
end.
```

Slăbiciunea acestui mod de rezolvare constă în faptul că dacă un element apare de mai mult de două ori, scrierea se repetă.

Scanned with OKEN Scanner

De exemplu, pentru datele de intrare (3,3,3) apare scris:

```
3 se repeta
3 se repeta
3 se repeta
```

Pentru a evita acest lucru, algoritmul se modifică puţin; anume, se consideră un element care nu există sigur în tablou, pe care îl folosim pentru „ştergere". De, câte ori se găsesc elemente egale, ele sunt înlocuite cu valoarea de ştergere.

```
var a:array[1..10] of integer;
    x,n,i:integer;
begin
     for n:=1 to 10 do readln(a[n]);
     write('dati un numar care nu este in tablou');
     readln(x);
     for n:=1 to 10 do
         if a[n]<>x then
            for i:=n+1 to 10 do
                if a[n]=a[i] then
                   begin
                        writeln(a[n], ' se repeta');
                        for j:=i to 10 do
                        if a[j]=a[i] then a[j]:=x
                   end
end.
```

2.3 (V)

Să se scrie un program care realizează intersecţia a două mulţimi de numere întregi date.

R 2.3

Programul citeşte cardinalul şi apoi elementele fiecărei mulţimi, validează datele de intrare astfel încât în niciuna dintre mulţimi să nu existe două elemente identice şi în final va realiza intersecţia.

```
program intersectie;
const m=10;
     {cardinalul maxim admis al unei multimi}
var a,b,c:array[1..m] of integer;
    na,nb,i,j,k:integer;
    este:boolean;
begin
     este:=false;
     write('cardinalul primei multimi:');
     readln(na);
     for i:=1 to na do
         repeat
                if este then writeln('element incorect');
                write('elementul ',i,' al multimii:');
                readln(a[i]);
                este:=false;
                for j:=1 to i-1 do
                    if a[i]=a[j] then este:=true
         until not este;
         ...
```


Scanned with OKEN Scanner

```
        {o secventa asemanatoare pentru sirul b cu nb elemente}
        ...
        k:=0;
        for i:=1 to na do
            begin
                 este:=false;
                 for j:=1 to nb do
                     if a[i]=b[j] then este:=true;
                 if este then
                    begin
                         k:=k+1;
                         c[k]:=a[i]
                    end
            end;
        write('Intersectia multimilor={');
        for i:=1 to k do write(c[i],',');
        writeln(#8,'}')
end.
```

2.4 (VI)

Să se determine printr-o singură parcurgere a unui şir dat de numere reale atât elementul cel mai mic cât şi elementul cel mai mare.

R 2.4

Algoritmul de determinare a minimului unui şir este foarte cunoscut: se iniţializează o variabilă (de exemplu min) cu valoarea primului element din şir ca valoare minimă potenţială, apoi se parcurge şirul în continuare actualizând valoarea variabilei *min* ori de câte ori un element din şir este mai mic decât minimul determinat până atunci. Alteori se iniţializează *min* nu cu primul element ci cu o valoare suficient de mare pentru a găsi cel puţin un element mai mic în şir (metoda iniţializării artficiale). Algoritmul propus aici pleacă de la prima dintre cele două metode şi foloseşte proprietatea evidentă că dacă un număr este mai mic decât cel mai mic dintre elementele anterioare (deci a ocupat postul de minim temporar), atunci el nu mai poate candida şi la funcţia de maxim al elementelor şirului.

```
program minimax;
type vector=array[1..100] of real;
var a:vector;
    i,n:byte;
    max,min:real;
begin
     writeln;
     write('Numarul elementelor din sir:');readln(n);
     for i:=1 to n do
         begin
              write('Elementul de rang ',i,':');
              read(a[i])
         end;
     min:=a[1];max:=a[1];
     for i:=2 to n do
         if a[i]<min then min:=a[i]        {*}
            else
                if a[i]>max then max:=a[i];
    writeln('Min=',min:6:3,' Max=',max:6:3)
end.
```


Scanned with OKEN Scanner

Observaţie: Dacă se apreciază că majoritatea elementelor din şir sunt în principiu crescătoare, atunci este mai eficient ca în locul instrucţiunii * să se folosească o instrucţiune „simetrică" din punct de vedere logic cu aceasta:

```
if a[i]>max then max:=a[i]
   else
       if a[i]<min then min:=a[i].
```

2.5 (V)

Se citeşte un şir cu un număr dat de componente numere reale şi se introduce apoi de la tastatură o valoare reală. Să se caute cât mai eficient şi să se stabilească dacă valoarea dată se găseşte sau nu în şir. Cum se poate îmbunătăţi algoritmul dacă se ştie cu certitudine că elementele date sunt în ordine crescătoare (sau descrescătoare) ?

R 2.5

Vom prezenta algoritmul de căutare secvenţială pentru a descrie cu acest prilej un artificiu de lucru cu şirurile prin care se pot evita testele de atingere a sfârşitului şirului. Iniţializând elementul imediat următor ultimului termen din şir cu valoarea căutată avem certitudinea că înaintarea prin şir se va opri la un moment dat fără să mai testăm indicele elementului la care s-a ajuns:

```
program cautare;
var a:array[1..1000] Of real;
    x:real;
    n,i:integer;
begin
      write('Nr.elemente=');
      readln(n);
      write('Introduceti elementele cu spatii sau enter:');
      for i:=1 to n do read(a[i]);
      readln;
      write('Numarul cautat='); readln( x);
      i:=1; a[n+1]:=x;
      while a[i]<>x do i:=i+1;
      if i=n+1 then writeln('nu este')
               else writeln('este')
end.
```

Dacă ştim că elementele şirului sunt ordonate (crescător de exemplu) vom putea adopta o altă strategie de căutare mai rapidă (cu ordin de complexitate logaritmic): metoda prin înjumătăţiri succesive ale intervalului de căutare, metodă care se mai numeşte **căutare binară**:

```
program cautare_binara;
var a:array[1..1000] Of real;
    i,n,margs,margd,jum:integer;
    x:real;
    este:boolean;
 begin
      write('Nr.elemente='); readln(n);
      write('Introduceti elementele cu spatii sau enter:');
```


Scanned with OKEN Scanner

```
    for i:=1 to n do read(a[i]); readln;
    write('Numarul cautat='); readln( x);
    if (x=a[1]) or (x=a[n]) then writeln('este')
    else
        begin
            margs:=1; margd:=n;
            {marginile stanga/dreapta ale intervalului}
            este:=false;
            while (margd-margs>1) and not este do
                begin
                    jum:=trunc((margs+margd)/2);
                    {mijlocul intervalului}
                    if  a[jum]=x then este:=true
                    else if x>a[jum] then margs:=jum
                                     else margd:=jum
                end;
            if este=true then writeln('este')
                         else writeln('nu e')
        end;
    readln
end.
```

2.6 (VIII)

Un tablou bidimensional cu *m* linii şi *n* coloane conţine numere întregi având modulul mai mic decât 100. Să se determine linia şi coloana unde se află elementul maximal al tabloului.

R 2.6

Citirea şi parcurgerea tabloului se fac în modul cunoscut (două structuri FOR imbricate), valoarea maximală a tabloului memorându-se în variabila *max*, poziţia ei reţinându-se în două variabile *lmax* şi *cmax*:

```
program max_tab;
const nmax=100;
type tablou=array[1..nmax,1..nmax]of integer;
var a:tablou;
    m,n,i,j:1..nmax;
    lmax,cmax:1..nmax;
    max:integer;
begin
   write('Numar linii=');readln(m);
   write('Numar coloane=');readln(n);
   writeln('Introduceti elementele pe linii,cu spatii intre ele:');
   for i:=1 to m do
       for j:=1 to n do read(a[i,j]);
   max:=a[1,1]; lmax:=1;cmax:=1;
   for i:=1 to m do
       for j:=1 to n do
           if a[i,j]>max then
              begin max:=a[i,j]; lmax:=i; cmax:=j end;
   writeln('Elem. max. se afla pe linia ',lmax,' si coloana ',cmax);
end.
```

Observaţie: Dacă tabloul conţine mai multe elemente egale, este posibil să detectăm valoarea maximală pe mai multe poziţii în tablou. Deci ne putem propune să determinăm poziţiile tuturor elementelor maximale:

Scanned with OKEN Scanner

```
program max_tab2;
const nmax=100;
type tablou=array[1..nmax,1..nmax]of integer;
var a:tablou;
    m,n,i,j:1..nmax;
    max:integer;
begin
   write('Numar linii=');readln(m);
   write('Numar coloane=');readln(n);
   max:=-1000;
   writeln('Introduceti elementele pe linii, cu spatii intre ele:');
   for i:=1 to m do
       for j:=1 to n do begin
           read(a[i,j]);
           if a[i,j]>max then max:=a[i,j]
       end;
   for i:=1 to m do
       for j:=1 to n do
           if a[i,j]=max then
              writeln('El.max. se afla pe linia ',i,' si col.',j);
end.
```

2.7 (VI)

Câte elemente ale unui tablou unidimensional sunt prime cu un număr dat?

R 2.7

Programul construit nu va defini un tablou; se va citi iniţial numărul şi apoi, odată cu citirea unui „element de tablou", va decide dacă el are sau nu divizori comuni cu valoarea dată. Avantajul acestei metode constă în eliminarea deciziei cu privire la dimensiunea maximă care trebuie stabilită la definirea tabloului. Se stabileşte dacă cele două numere sunt prime între ele cu ajutorul algoritmului lui Euclid, varianta cu scăderi.

```
program prim;
var a,b,c,d,k,n,i:integer;
begin
     write('Dati numarul de elemente ale vectorului: n=');readln(n);
     write('Dati numarul de comparat: b='); readln(b);
     writeln('Dati elementele vectorului:');
     k:=0; c:=b;
     for i:=1 to n do begin
         read(a); d:=a;
         repeat
               if d<c then c:=c-d else d:=d-c
         until (d=c);
         if d=1 then begin
            writeln('Numarul a[',i,']=',a,' este prim cu ',b);
            k:=k+1
         end;
         c:=b;
     end;
     if k=0 then writeln('Nu sunt numere prime cu ',b)
            else writeln('In total sunt ',k,' numere prime cu ',b);
end.
```


Scanned with OKEN Scanner

2.8 (VI)

Fiind date elementele unui şir de numere reale, se cere să se tipărească şirul obţinut din acesta intercalând între oricare două elemente media lor aritmetică.

Exemplu: Pentru şirul

```
7  4.2  11  -1  20   se obţine şirul
7  5.6  4.2  7.6  11  5  -1  9.5  20
```

R 2.8

În loc să citim elementele într-un vector x, pe care să le transferăm ulterior într-un vector y cu intercalarea valorilor medii după algoritmul:

```
j:=1;
for i:=1 to n-1 do begin
    y[j]:=x[i];
    y[j+1]:=(x[i]+x[i+1])/2)/2;
    inc(j,2)
end;
y[j]:=x[n],
```

Sugerăm citirea şi construirea lui direct în vectorul y:

```
readln(y[1]);
for i:=2 to n do begin
    readln(y[2*i-1]);
    y[2*i-2]:=(y[2*i-3]+y[2*i-1])/2
end;
```

2.9 (IX)

Să se afle cea mai mare diferenţă dintre doi termeni consecutivi ai unui tablou.

R 2.9

Se calculează toate diferenţele posibile dintre termenii consecutivi şi se reţine valoarea maximă a acestor diferenţe. Trebuie atrasă atenţia asupra modului în care se realizează ca diferenţa calculată să fie totdeauna pozitivă: se foloseşte funcţia abs, care este definită matematic: $\text{abs}(a) = \begin{cases} a \text{ dacă } a \geq 0 \\ -a \text{ dacă } a < 0 \end{cases}$.

```
program diferenta;
var a:array[1..30] of integer;
    b,n,i,k:integer;
begin
     repeat
          write('Dati nr. de elemente din vector: n='); readln(n);
     until (n>1)and(n<=30);
     writeln('Dati elementele vectorului:');
     for i:=1 to n do read(a[i]);
     b:=abs(a[2]-a[1]); k:=2;
     for i:=2 to n do
         if b<abs(a[i]-a[i-1]) then begin
            b:=abs(a[i]-a[i-1]);k:=i
         end;
     writeln('Cea mai mare diferenta este ',b);
     writeln('Ea apare intre termenii ',k-1,' si ',k);
end.
```


Scanned with OKEN Scanner

2.10 (VIII)

Definim factorul de „aglomerare" al unei populaţii pe o suprafaţă dată ca fiind numărul maxim de vecini pe care îi are un membru al acelei populaţii (prin vecin înţelegând un alt membru al acelei populaţii, aflat în imediata sa apropiere pe oricare din direcţiile orizontală, verticală sau diagonală). Se dă distribuţia membrilor populaţiei printr-un tablou cu m linii şi n coloane în care un membru al populaţiei este reprezentat printr-un element 1, iar o poziţie liberă (neocupată) a suprafeţei considerate printr-un element 0. Să se determine factorul de aglomerare.

R 2.10

Esenţa problemei constă din numărarea vecinilor fiecărui element reţinând numărul maxim. Testarea existenţei vecinilor în imediata apropiere a elementului 1 de pe poziţia (i, j) se rezumă la compararea elementelor de pe poziţiile $(i, j+1)$, $(i-1, j+1)$, $(i-1, j)$, $(i-1, j-1)$, $(i, j-1)$, $(i+1, j-1)$, $(i+1, j)$, $(i+1, j+1)$ cu numărul 1. Realizând o succesiune de 8 structuri if sau cu o structură de tipul:

```
for x:=-1 to 1 do
   for y:=-1 to 1 do
     if (x<>0) or (y<>0) do
        if t[i+x,j+y]=1 then nr_vec:=nr_vec+1
```

(care este mai „compactă", dar nu este mai eficientă prin aceasta, adică numărul de comparări a crescut inutil), putem considera că problema numărării vecinilor elementului t[i, j] este rezolvată. Ce facem însă cu elementele de pe marginea tabloului care nu au dintre cei 8 vecini decât 3 sau 5?

Recomandăm, în locul condiţiilor compuse de tipul:

```
if (i>1) and (j<n) and (t[i-1,j+1]=0) then nr_vec:=nr_vec+1
```

un artificiu de iniţializare artificială prin bordarea tabloului cu elemente egale cu 0 (elemente a căror „consultare" să nu influenţeze numărul de vecini ai elementelor de pe margini).

S-au mai folosit doi vectori „de deplasament", x şi y cu ajutorul cărora se îmbină eficienţa celor 8 instrucţiuni if şi eleganţa alternativei a doua (dezavantajul acestei variante – a treia – fiind numai consumul suplimentar de memorie):

```
program aglomerare;
      {elementele x[i],y[i] constituie deplasamentele pe verticala
      respectiv orizontala ale uneia dintre cele 8 pozitii vecine}
const x:array[1..8]of -1..1=(0,-1,-1,-1,0,1,1,1);
      y:array[1..8]of -1..1=(1,1,0,-1,-1,-1,0,1);
type tablou=array[0..41,0..41] of 0..1; {indicii 0 si 41 permit
        bordarea unui tablou cu indicii de la 1 la cel mult 40}
var t:tablou;
    m,n,i,j,k:byte;
    nr_vec,vec_max:0..8;
begin
     write('Numarul de linii:');readln(m);
     write('Numarul de coloane:');readln(n);
     writeln('Dati cele ',m*n,' elemente pe linii:');
     {se pot introduce oricum, despartite prin oricate spatii sau
      <CR> din cauza folosirii procedurii read in loc de readln}
```


Scanned with OKEN Scanner

```
    for i:=1 to m do
        for j:=1 to n do read(t[i,j]); readln;
    for i:=0 to m+1 do      {borduri laterale}
        begin t[i,0]:=0; t[i,n+1]:=0 end;
    for j:=0 to n+1 do      {borduri orizontale}
        begin t[0,j]:=0; t[m+1,j]:=0 end;
    for i:=1 to n do begin  {tiparirea tabloului cu caractere}
        for j:=1 to n do
        if t[i,j]=1 then write('* ') else write('. ');
        writeln
    end;
    vec_max:=0;
    for i:=1 to n do
        for j:=1 to n do
            if t[i,j]=1 then begin
               nr_vec:=0;
               for k:=1 to 8 do
                   if t[i+x[k],j+y[k]]=1 then nr_vec:=nr_vec+1;
               if nr_vec>vec_max then vec_max:=nr_vec
            end;
    writeln('Factorul de aglomerare este:',vec_max);
end.
```

2.11 (IX – C)

Se citesc elementele – numere întregi – ale unui tablou pătrat de dimensiune n ($n \leq 20$). Să se afişeze şirul format prin parcurgerea tabloului în spirală, din colţul din stânga – sus către dreapta până în centrul tabloului.

Exemplu: Pentru tabloul de dimensiune 4

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

se va afişa şirul 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10.

R 2.11

Imaginăm secvenţa de 4 cicluri for prin care parcurgem un singur inel al spiralei (cel mai exterior), apoi generalizăm formula pentru toate cele $(n + 1)$ div 2 inele:

```
program citire_in_spirala;
uses crt;
var k,n,i,j:integer;
    a:array[1..20,1..20]of integer;
begin
    clrscr;
    write('Dimensiunile tabloului:');readln(n);
    writeln('Elementele tabloului pe linii:');
    for i:=1 to n do
        for j:=1 to n do read(a[i,j]);
    for k:=1 to (n+1) div 2 do begin
        for i:=k to n-k+1 do write(a[k,i],' ');
        for j:=k+1 to n-k+1 do write(a[j,n-k+1],' ');
        for i:=n-k downto k do write(a[n-k+1,i],' ');
        for j:=n-k downto k+1 do write(a[j,k],' ')
    end
end.
```


Scanned with OKEN Scanner

2.12 (VII)

Un tablou dat cu m linii şi n coloane ($m, n \leq 40$) este format numai din numere de o singură cifră. Să se afişeze toate „ramele centrate" succesive ce se pot forma cu elementele tabloului.

Exemplu: Pentru tabloul cu 5 linii şi 6 coloane:

```
1 2 3 4 5 6
2 3 4 5 6 7
3 4 5 6 7 8
4 5 6 7 8 9
5 6 7 8 9 0
```

se vor afişa

```
1 2 3 4 5 6
2         7
3         8
4         9
5 6 7 8 9 0
3 4 5 6
4     7
5 6 7 8
5 6
```

(această ultimă ramă plină fiind considerată ramă degenerată)

R 2.12

Raţionamentul este asemănător cu cel din problema precedentă, singura problemă ivindu-se la ramele interioare degenerate ca şi cea din exemplu (provenite din tablouri cu un număr impar de linii sau coloane) unde rama nu se mai obţine prin utilizarea celor 4 instrucţiuni for:

```
program rame;
var a:array[1..40,1..40] of byte;
    m,n,i,j,k,nr_rame:byte;
begin
     write('nr.linii=');readln(m);
     write('nr.coloane=');readln(n);
     for i:=1 to m do
         for j:=1 to n do begin
             write('a[',i,',',j,']='); readln(a[i,j])
         end;
     if m<n then nr_rame:=(m+1) div 2
            else nr_rame:=(n+1) div 2;
     for j:=1 to nr_rame do begin
         for k:=j to n-j+1 do write(a[j,k]:4); writeln;
         if j<>n-j+1 then
            for k:=j+1 to m-j do
                writeln(a[k,j]:4,'':4*(n-2*j),a[k,n-j+1]:4)
         else for k:=j+1 to m-j do writeln(a[k,j]:4);
         if m-j+1<>j then
            for k:=j to n-j+1 do write(a[m-j+1,k]:4);readln
     end
end.
```

2.13 (VI – C)

Se citesc de la tastatură două numere naturale de cel mult 4 cifre. Să se afişeze valoarea zecimală exactă a raportului lor, folosind un număr minim de caractere.

Scanned with OKEN Scanner

R 2.13

Cunoscând algoritmul împărţirii numerelor naturale (clasa a V-a) vom simula împărţirea prin program, coborând zecimalele deîmpărţitului (zerouri) cifră cu cifră. Înţelegând prin cerinţa „număr minim de caractere" că pentru numerele zecimale infinite (obligatoriu periodice deoarece provin din fracţii de numere întregi, deci din numere raţionale) se va folosi reprezentarea lor cu perioada principală între paranteze, vom avea grijă să memorăm resturile parţiale ale împărţirilor pentru a sesiza când se repetă unul dintre resturi.

```
program raport;
var poz_repetare,i,j,k,a,b:integer;
    c:array[1..10000] of byte;
    r:array[1..10000] of integer;
begin
     write('introduceti a,b:'); readln(a,b);
     if b=0 then begin
        writeln('Cine a mai vazut raport cu numitorul 0?'); halt
     end;
     write(a div b);
     if a mod b=0 then writeln
     else begin
          write('.'); i:=0; poz_repetare:=0;
          repeat
                a:=a mod b;                {Restul momentan}
                for j:=1 to i do
                    if r[j]=a then poz_repetare:=j;
                i:=i+1;
                r[i]:=a;                   {memorarea restului}
                a:=a*10;                   {coborarea unui zero}
                c[i]:=a div b {cifra curenta a rezultatului}
          until (a=0) or (poz_repetare<>0);{pana se imparte
              exact sau se obtine un rest care a mai aparut}
          if poz_repetare<>0 then begin
             for k:=1 to poz_repetare-1 do write(c[k]);
             write('(');
             for k:=poz_repetare to i-1 do write (c[k]);
             writeln(')')
          end
          else for k:=1 to i do write(c[k])
     end
end.
```

2.14 (X – C)

Se citesc două şiruri de numere naturale, fiecare şir citindu-se până la întâlnirea numărului 13 (număr care nu se consideră ca făcând parte din şir). Să se stabilească dacă cele două şiruri formează două mulţimi de numere direct proporţionale, invers proporţionale, neproporţionale sau între ele nu se poate stabili proporţionalitate.

Exemplu: se citesc 12 80 40 20 4 13 şi 100 25 15 5 50 13.

Mulţimile 12 80 40 20 4 şi 100 25 15 5 50 reprezintă mărimi direct proporţionale deoarece 12 / 15 = 80 / 100 = 40 / 50 = 20 / 25 = 4 / 5 numărătorii fiind din prima mulţime, iar numitorii din a doua.

Scanned with OKEN Scanner

R 2.14

A stabili proporţionalitatea înseamnă a decide dacă se pot forma cu toate numerele din cele două şiruri perechi (o pereche fiind formată dintr-un element al primului şir şi un element al celui de-al doilea şir) astfel încât rapoartele (produsele – pentru proporţionalitatea indirectă) elementelor ce formează perechile să fie egale. Evident, dacă nu există un acelaşi număr de elemente, nu se poate vorbi despre proporţionalitate. O metodă ar fi să împerechem în toate modurile posibile elementele celor două mulţimi şi să verificăm dacă toate perechile au fie raportul, fie produsul constant. Însă am avea practic de generat toate permutările unei mulţimi pentru a forma perechi cu elementele celeilalte mulţimi, ceea ce înseamnă un algoritm exponenţial laborios.

Observând proprietăţile matematice ale numerelor care formează un şir de rapoarte egale, vom constata că dacă numitorii sunt în ordine crescătoare, atunci şi numărătorii sunt tot în ordine crescătoare; pentru şir de produse egale, dacă elementele din prima mulţime sunt în ordine crescătoare, atunci cele din a doua mulţime sunt în ordine descrescătoare. Deci ordonăm elementele fiecărei mulţimi şi formăm perechi de elemente corespunzătoare pentru a verifica proporţionalitatea directă şi perechi de elemente simetric corespunzătoare (primul cu ultimul, al doilea cu penultimul etc.) pentru a verifica proporţionalitatea indirectă:

```
program multimi_proportionale;
var a,b:array[1..1000] of word;
    na,nb,aux,i,j:word;
    rap,prod:real;
    gata,dir_prop,inv_prop:boolean;
begin
     write('Introduceti elem. multimii A (13-->sfarsit multime):');
     na:=0; read(aux);
     while aux<>13 do
           begin
                inc(na); a[na]:=aux; read(aux)
           end;
     readln; write('A={');
     for i:=1 to na do write(a[i],',');
     writeln(#8,'}');
     { .... Idem pentru multimea B ...}
     if na<>nb then
        writeln('Nu se poate stabili proportionalitate')
     else
         begin {ordonarea elementelor multimii a}
              i:=0;
              repeat
                    i:=i+1;        {a cata parcurgere se executa}
                    gata:=true;
                    for j:=1 to na-i do      {parcurgerea sirului}
                        if a[j]>a[j+1] then
                           begin         {nterschimbarea elementelor}
                                aux:=a[j];a[j]:=a[j+1];a[j+1]:=aux;
                                gata:=false
                           end;
              until gata;
```


Scanned with OKEN Scanner

```
        { ... La fel ordonarea elementelor multimii b ... }
        rap:=a[1]/b[1];prod:=a[1]*b[nb];
        dir_prop:=true;inv_prop:=true;
        for i:=2 to nb do
            begin
                if a[i]/b[i]<>rap then dir_prop:=false;
                if a[i]*b[nb-i+1]<>prod then inv_prop:=false
            end;
        if dir_prop then writeln('Direct proportionale');
        if inv_prop then writeln('Invers proportionale');
        if not dir_prop and not inv_prop then
           writeln('Neproportionale')
    end
end.
```

Observaţie: Pentru ca programul de mai sus să poată fi rulat trebuie să adaugaţi în locul comentariilor {... ...} blocuri de program copiate şi modificate corespunzător pentru mulţimea B.

2.15 (IX)

Se citesc elementele unui tablou pătrat. Să se interschimbe liniile şi coloanele tabloului astfel încât, în tabloul final, elementele de pe diagonala principală să fie în ordine descrescătoare. Să se tipărească atât tabloul iniţial cât şi cel obţinut.

Exemplu: pentru tabloul cu 4 linii şi 4 coloane:

```
1 3 8 2        se obţine:        9 1 5 4
0 1 0 5                          5 7 2 2
4 5 9 1                          8 2 3 1
2 2 5 7                          0 5 1 0
```

R 2.15

Prin permutări de linii şi/sau coloane, se aduce în colţul din stânga sus (poziţia (1, 1)) cel mai mare element al tabloului, apoi pe poziţia (2, 2) se aduce elementul maximal al tabloului obţinut prin eliminarea primei linii şi coloane, etc.

```
program matrice;
const dmax=20;
var a:array[1..dmax,1..dmax] of integer;
    n,i,j,k,lmax,cmax:byte; max,x:integer;
begin
    write('Dati dimensiunea matricii:');readln(n);
    for i:=1 to n do
        for j:=1 to n do
            begin
              write('a[',i,',',j,']='); readln(a[i,j])
            end;
    for i:=1 to n do      {scrierea tabloului initial}
        begin
            for j:=1 to n do write(a[i,j]:4); writeln
        end;
    for k:=1 to n-1 do { k=indicele elementului de pe diagonala}
        begin
           max:=a[k,k];lmax:=k;cmax:=k; {determinarea maximului}
           for i:=k to n do
              for j:=k to n do
                 if a[i,j]>max then
                     begin max:=a[i,j]; lmax:=i; cmax:=j end;
```


Scanned with OKEN Scanner

```
                {interschimba linia maximului cu linia k}
                for j:=1 to n do
                    begin
                      x:=a[k,j]; a[k,j]:=a[lmax,j]; a[lmax,j]:=x
                    end;
                {interschimba coloana maximului cu coloana k}
                for i:=1 to n do
                    begin
                      x:=a[i,k]; a[i,k]:=a[i,cmax]; a[i,cmax]:=x
                    end
          end;
      writeln;
      for i:=1 to n do
          begin
                for j:=1 to n do write(a[i,j]:4); writeln
          end;
      writeln
end.
```

2.16 (X)

Se dă o matrice. Să se determine cea mai mică linie a sa din punct de vedere lexicografic.

R 2.16

Fiind dați doi vectori $a = (a_1, a_2, .., a_k)$ și $b = (b_1, b_2, .., b_k)$, spunem că $a < b$ dacă există un indice i astfel încât $a_i < b_i$ și $a_j = b_j$ pentru orice $j < i$.

```
program lexicogr;
var a:array[1..10,1..10] of integer;
    min:array[1..10] of integer;
    i,j,k,n,m:integer;
    flag:boolean;
begin
      write('Dati dimensiunile matricii (linii,coloane): ');
      readln(n,m);
      writeln('Dati elementele matricii pe linii:');
      for i:=1 to n do
          for j:=1 to m do read(a[i,j]);
      for j:=1 to m do min[j]:=a[1,j];
      for i:=1 to n do
          begin
                j:=1;
                while min[j]=a[i,j] do j:=j+1;
                if (j<=m) and (min[j]>a[i,j]) then
                   for k:=1 to m do min[k]:=a[i,k];
          end;
      writeln('Linia minima lexicografic din matrice este:');
      for j:=1 to m do write(min[j],' ');
end.
```

2.17 (V – C, Concurs clasa a V-a, Palatul Copiilor, 1995)

N pitici așezați unul în spatele celuilalt, poartă căciuli colorate roșii sau albe. Fiecare pitic spune două numere, primul reprezentând numărul de căciuli albe, celălalt numărul de căciuli roșii pe care le poartă piticii din fața sa.

Scanned with OKEN Scanner

a) Ştiind că piticii cu căciulă roşie mint (dau incorect cel puţin unul dintre cele două numere), iar cei cu căciulă albă spun întotdeauna adevărul, să se determine culoarea căciulii fiecărui pitic.

Se vor citi de la tastatură:

– numărul n de pitici ($n \leq 100$)

– cele n perechi de numere.

Se va tipări pe ecran o succesiune de litere *A* şi *R* reprezentând culorile alb, respectiv roşu ale căciulilor în ordinea în care stau piticii în şir.

b) Ştiind că fiecare pitic îşi păstrează culoarea căciulii determinată la punctul (a), să se afle dacă este posibilă schimbarea ordinii piticilor în şir astfel încât toţi piticii să spună adevărul. În caz afirmativ se vor tipări numerele de ordine iniţiale ale piticilor în noua ordine stabilită.

Exemplu:

Pentru datele de intrare:

```
     n:      5
pitic1:    2 1
pitic2:    0 1
pitic3:    1 1
pitic4:    0 0
pitic5:    2 2
```

Se obţin rezultatele:

a) R A A R A

b) DA : 4 2 3 1 5

R 2.17

a) Rezolvarea problemei nu prezintă nici un fel de dificultate, ea constând din analiza succesivă a „afirmaţiilor" piticilor după algoritmul de principiu:

```
Citeste N
Pentru i=1,n
  Citeste a,r
  Daca spune_adevarul
       atunci este_alba
       altfel este_rosie
```

b) Un algoritm de rezolvare presupune determinarea pe rând a perechilor de numere care au suma (adică numărul de pitici din faţă) 0, apoi 1, apoi 2, .. , până la $n - 1$, condiţia de existenţă a soluţiei fiind ca la fiecare pas să existe o astfel de pereche şi piticul corespunzător să spună adevărul.

```
i:=0; tot_albe=0; tot_rosii=0
Repeta
  Cauta_pitic pentru care r+a=i si spune_adevarul
  Daca exista
       atunci Daca are_caciula_rosie atunci tot_rosii=tot_rosii+1
                                      altfel tot_albe=tot_albe+1
       altfel nu_exista
  i:=i+1
pana cand nu_exista sau i=n
```


Scanned with OKEN Scanner

```
program pitici;
var n,i,j,k,n_a,n_r:byte;
    sir:array[1..100] of char;
    a,r:array[1..100] of byte;
begin
     write('Nr.pitici=');readln(n);
     n_a:=0; n_r:=0; {linie de initializari inutila in TURBO 7.0}
     for i:=1 to n  do
         begin
              write('-pitic-',i:2,'=');readln(a[i],r[i]);
              if (n_a=a[i]) and (n_r=r[i]) then
                 begin n_a:=n_a+1; sir[i]:='A' end
              else
                  begin n_r:=n_r+1; sir[i]:='R' end
         end;
     write('a) ');
     for i:=1 to n do write(sir[i]:2); writeln;
     n_a:=0;n_r:=0;i:=0; write('b) ');
     repeat
           j:=0;
           for k:=1 to n do
               if (a[k]+r[k]=i)and(n_a=a[k])and(n_r=r[k]) then j:=k;
           write(j:3);
           if j<>0 then
               if sir[j]='A' then n_a:=n_a+1
                             else n_r:=n_r+1;
           i:=i+1;
     until (j=0) or (i=n);
     if j=0 then writeln('b) Nu exista solutie')
            else writeln
end.
```

2.18 (VIII – C, Concurs clasa a VIII-a, Palatul Copiilor, 1995)

Se introduc de la tastatură n ($n \leq 10000$) numere întregi.

Să se genereze şi să se tipărească un tablou bidimensional care conţine toate numerele date, fiecare o singură dată, tablou care să aibă următoarele proprietăţi:

– elementele de pe fiecare linie şi coloană să fie ordonate crescător

– perimetrul tabloului să fie minim.

Exemplu: Pentru datele de intrare:

```
4
5 8 0 9
```

ieşirea va fi:

```
0 5
8 9
```

R 2.18

O primă problemă ar fi determinarea dimensiunilor tabloului. Din condiţia de minim impusă asupra perimetrului, deducem că tabloul ar trebui să fie „cât mai pătrat", adică diferenţa dintre numărul de linii şi de coloane să fie cât mai mică. Vom determina un divizor d al lui n cât mai apropiat de $\sqrt{n}$. Atunci d va reprezenta numărul de linii, iar n/d va reprezenta numărul de coloane (sau invers).

Scanned with OKEN Scanner

Pentru cerinţa ca elementele de pe linii şi coloane să fie ordonate crescător este suficient dacă, de exemplu, ordonăm crescător toate elementele şi le aşezăm în tablou pe linii de la stânga la dreapta (sau pe coloane, de sus în jos).

```
program tablou_perim_minim;
var a:array[1..10000] of integer;
    n,i,j,aux,dim:integer;
begin
     write('Dati numarul de componente:');readln(n);
     write('Dati componentele:');
     for i:=1 to n do read(a[i]);
     for i:=1 to n-1 do
         for j:=1 to n-i do
             if a[j]>a[j+1] then begin
                aux:=a[j]; a[j]:=a[j+1]; a[j+1]:=aux
             end;
     dim:=round(sqrt(n));
     while n mod dim<>0 do dim:=dim+1;
     for i:=1 to n div dim do begin
         for j:=1 to dim do write(a[(i-1)*dim+j]:4);writeln
     end
end.
```

2.19 (VIII)

La o adunare a oamenilor de afaceri s-a propus înfiinţarea de către un grup de investitori a unei societăţi comerciale cu un capital de pornire c, dat. Cunoscând suma de care dispune fiecare dintre cei n oameni de afaceri, să se aleagă acele persoane care totalizează suficient capital pentru a putea înfiinţa societatea şi în acelaşi timp să formeze un grup cât mai restrâns (pentru ca posibilităţile de a interveni neînţelegeri între ei să fie cât mai mici).

R 2.19

Enunţul este o variantă a unei probleme clasice (problema rucsacului), în care trebuie să alegem dintr-un vector dat s un număr minim de componente care au suma peste o limită dată. Pentru aceasta, vectorul sumelor deţinute de afacerişti se ordonează descrescător şi se introduc elemente în soluţie, începând cu primul element, până ce sumele alese reuşesc să depăşească pe c (capitalul obligatoriu de pornire).

```
program oameni_de_afaceri;
type vector=array[1..600] of longint;
var n,i,j:integer;  s:vector;
    suma,c,t:longint;
begin
{..........citirea datelor,ordonarea vectorului s........}
     suma:=0; i:=1;
     while  (suma<=c)and (i<=n) do begin
             write(s[i]:10); suma:=s[i]+suma;
             i:=i+1
     end;
     if i>n then writeln('-->nu sunt suficiente sumele disponible')
end.
```


Scanned with OKEN Scanner

2.20 (VIII – C, Problemă propusă, Balcaniada de Informatică, Salonic, 1994)

Fiind dat un set de segmente (intervale deschise) situate pe aceeaşi axă, se cere colorarea lor cu un număr minim de culori astfel încât orice două segmente care se intersectează să aibă culori diferite. Fiecare interval este dat pe o linie sub forma unei perechi de numere întregi din intervalul [1,1000]; primul număr este totdeauna mai mic decât al doilea. Pe prima linie este precizat numărul de segmente.

Exemplu: pentru intrarea

```
4
5 10
100 200
6 30
1 1000
```

ieşirea va fi:

```
3
```

R 2.20

Soluţia este foarte simplă şi se reduce la determinarea numărului maxim de segmente care se suprapun. Axa pe care sunt scrise segmentele este un tablou linie. Elementele lui sunt iniţial 0 (nu există nici un segment); fiecare segment se colorează adunând elementele sale din linie cu 1. Dacă de exemplu există un element *k* al liniei cu *linie[k]:=4*, aceasta înseamnă că *linie[k]* este componentă a 4 segmente, deci aceste segmente necesită patru culori diferite pentru a fi distinse.

```
program colorare;
var linie: array[1..1000] of integer;
    a,b,max,i,j,n:integer;
begin
     write('dati numarul de segmente. n='); readln(n);
     for i:=1 to 1000 do linie[i]:=0;
     for i:=1 to n do begin
         readln(a,b);
         for j:=a+1 to b do linie[j]:=linie[j]+1
     end;
     max:=0;
     for i:=1 to 1000 do
         if max<linie[i] then max:=linie[i];
     writeln('Numarul maxim de culori necesare este ',max)
end.
```

2.21 (VI)

Se generează un număr aleator de patru cifre distincte. Să se ghicească numărul prin încercări: se propune de la tastatură un număr de 4 cifre distincte şi calculatorul precizează câte dintre cifrele numărului propus sunt în numărul căutat chiar pe locul lor (centrate) şi câte dintre ele sunt în numărul căutat dar pe altă poziţie (necentrate). Execuţia se încheie când se propune chiar numărul generat iniţial (răspuns: 4 centrate).

R 2.21

Se poate genera fiecare cifră a numărului *n* cu ajutorul funcţiei *random* (*1+random(9)=prima* şi *random(10)=urmatoarele*), controlând cu ajutorul unui vector *f* de valori logice dacă nu cumva o cifră fixată se repetă:

Scanned with OKEN Scanner

```
for i:=0 to 9 do f[i]:=false;
randomize; n[1]:=1+random(9);
f[n[1]]:=true;
for i:=2 to 4 do begin
    repeat n[i]:=random(10) until not f[n[i]];
    f[n[i]]:=true
end;
```

Ghicirea se desfăşoară conform algoritmului:

```
repeat
     write('Numarul propus=');
     for i:=1 to 4 do begin
         read(ch);              {un caracter cifra}
         p[i]:=ord(ch)-ord('0')
     end;
     readln;
     c:=0;nc:=0;
     for i:=1 to 4 do
         for j:=1 to 4 do
             if n[i]=p[j] then
                 if i=j then c:=c+1
                        else nc:=nc+1;
     writeln('centrate=',c,#13#10,'necentrate=',nc)
until c=4
```

Probleme propuse

2.22 (VI)

Pentru *n* numere date se calculează media aritmetică şi două valori procentuale cu o zecimală: procentul de numere dintre cele date care sunt strict mai mici decât media aritmetică a întregului şir şi procentul de numere care sunt strict mai mari decât media aritmetică. În ce caz suma celor două procente este 1 (100%) ?

Exemplu: Pentru şirul 3 20 11 56 se obţin procentele

```
75.0% (elemente mai mici decât 22.5)
25.0% (elemente mai mari decât 22.5)
```

2.23 (IX)

Să se tipărească toate permutările circulare ale unui vector de numere reale dat. De exemplu o permutare circulară a şirului 5,7,25,8,-1,30,2 este şirul 8,-1,30,2,5,7,25.

2.24 (VI)

Într-un grup de *n* persoane, fiecare îşi face publică vârsta spunând două numere: ani şi luni. Să se afişeze care este vârsta pe care o au cele mai multe persoane din grup. Se cere o singură soluţie.

2.25 (X – C)

Cunoscându-se lungimea unui şir şi componentele sale, să se tipărească o secvenţă de lungime maximă care formează o progresie aritmetică. Prin secvenţă

Scanned with OKEN Scanner

de lungime k înţelegem componentele şirului dat pornind de la un indice oarecare i până la indicele $i + k - 1$.

Exemplu: pentru şirul 5 2 15 23 2 4 6 –1 33.5 81 21 –19
sunt răspunsuri corecte 2 4 6 şi 81 21 –19

2.26 (V)

Se citesc n numere naturale. Să se determine din câte secvenţe monotone este format şirul dat şi să se tipărească fiecare secvenţă. Prin secvenţă monotonă se înţelege un şir de termeni consecutivi care sunt, fie în ordine crescătoare, fie în ordine descrescătoare.

Exemplu: şirul 2 8 12 4 20 24 30 5 3 1 1 0 este format din 3 secvenţe monotone şi anume (2 8 12), (4 20 24 30), (5 3 1 1 0) (primele două fiind crescătoare iar cea de-a treia – descrescătoare).

2.27 (IX)

Se citesc de la tastatură numărul de linii şi coloane ale unui tablou şi apoi elementele acestuia, citite pe linii. Să se determine „punctele şa" ale tabloului precizând poziţia fiecăruia (linie-coloană). Se numeşte „punct şa" acel element care este minim pe linia sa şi maxim pe coloana sa.

2.28 (VI)

Să se scrie un program care să realizeze reuniunea a două mulţimi.

2.29 (VI)

Se dă un tablou cu n elemente numere întregi. Să se determine toate tripletele crescătoare de elemente consecutive.

2.30 (XI)

Să se determine rangul unei matrici.

Indicaţie: Folosind regulile:

– se pot permuta două linii sau două coloane;

– se poate înmulţi o linie sau o coloană cu o constantă nenulă;

– se poate aduna la o linie (coloană) o altă linie (respectiv coloană) înmulţită cu o contantă; se fac în matrice cât mai multe elemente zero. Când nu se mai pot obţine elemente nule, se numără elemnentele nenule; numărul lor dă rangul matricii.

2.31 (XI)

Să se rezolve un sistem liniar de: a) două ecuaţii cu două necunoscute;
b) trei ecuaţii cu trei necunoscute.

2.32 (XI)

Se citesc două matrici pătrate A şi B de dimensiuni $n \times n$ cu elemente numere reale. Să se calculeze A*B–B*A.

Indicaţie: dacă C=A*B, atunci c_{ij} se calculează cu următoarea structură

```
c[i,j]:=0
for k:=1 to n do c[i,j]:=c[i,j]+a[i,k]*b[k,j]
```

2.33 (XII)

Se dă un tablou bidimensional $n*n$ care reprezintă o lege de compoziţie pe mulţimea $\{1, 2, ..., n\}$. Să se determine dacă legea este:

a) comutativă; b) are element neutru;

c) fiecare element admite invers.

Scanned with OKEN Scanner

III. ŞIRURI DE CARACTERE ŞI MULŢIMI

Probleme propuse

3.1 (V)

Să se scrie toate prefixele unui cuvânt dat; prin prefix se înţelege şirul format din primele caractere ale cuvântului (minim un caracter, maxim toate).

R 3.1

Funcţia *length(s:string)* întoarce lungimea efectivă a şirului *s* şi este echivalentă cu *ord(s[0])* (folosind helpul, cercetaţi funcţia *ord* şi inversa ei – *chr*). Funcţia *copy(s,început,lung)* întoarce şirul format din primele *lung* caractere ale lui *s* începând de la poziţia *început*.

```
program prefixe;
var s:string; i:byte;
begin
     write('Dati cuvantul: ');readln(s);
     for i:=1 to length(s) do
         writeln(' Prefix de lungime ',i:3,': ', copy(s,1,i))
end.
```

3.2 (VI)

Se dă un text de maxim 30 caractere. Să se listeze toate cuvintele de două caractere din acest text.

R 3.2

Să presupunem că oricare două cuvinte consecutive din text sunt separate printr-un spaţiu şi numai unul. Programul va parcurge tot textul, verificând la fiecare pas patru caractere consecutive: dacă în acest grup primul şi al patrulea caracter sunt spaţii, pe poziţiile 2 şi 3 se află un cuvânt de două litere, care se listează.

```
program doua-litere;
var a:string[30];
    n:integer;
begin
   readln(a);
   for n:=2 to length(a)-2 do
   if (a[n-1]=' ')and(a[n]<>' ')and(a[n+1]<>' ')and(a[n+2]=' ') then
      writeln(copy(a,n,2))
end.
```

3.3 (VI)

Se dă un număr *n* ($n > 0$); să se scrie cel mai mare număr ce se poate forma cu cifrele distincte ale lui *n*.

R 3.3

Algoritmul folosit citeşte *n* şi-l „sparge" în cifre; introduce apoi fiecare cifră în variabila *a* de tip mulţime; proprietatea esenţială a tipului mulţime este aceea că ea reţine numai cifrele distincte. În faza a doua, se formează numărul cu elementele lui *a* luate descrescător. Facem observaţia că operaţia + dintre două mulţimi reprezintă reuniunea, iar relaţia *in* desemnează apartenenţa.

Scanned with OKEN Scanner

```
program numar;
var a:set of 0..9;
    i,j:integer;
    x,n:longint;
begin
     write('dati numarul n=');readln(n);
     a:=[]; x:=n;
     while x>0 do
           begin a:=a+[x mod 10]; x:=x div 10 end;
     x:=0;
     for i:=9 downto 0 do
     if i in a then x:=10*x+i;
     writeln('Cel mai mare nr.cu cifre diferite ale lui ',n,' este ',x);
end.
```

3.4 (VII)

Se dă un text. Să se decidă dacă textul foloseşte numai caractere alfanumerice.

Observaţie: prin caractere alfanumerice înţelegem de regulă mulţimea formată din litere şi cifre.

R 3.4

Se selectează în variabila *b* de tip mulţime toate caracterele folosite de text. Apoi se verifică dacă mulţimea astfel obţinută este inclusă în mulţimea *a* formată din toate caracterele alfanumerice (operaţia de incluziune se notează <=).

```
program alfa;
const a=['0'..'9','a'..'z','A'..'Z'];
var b:set of char;
    x:string;
    i:integer;
begin
     b:=[];
     writeln('Dati textul:'); readln(x);
     for i:=1 to length(x) do b:=b+[x[i]];
     if b<=a then begin
        writeln('Textul foloseste numai caractere alfanumerice.');
        writeln('Acestea sunt:');
        for i:=48 to 122 do
            if chr(i) in b then write(chr(i),'  ');
        writeln
     end
     else begin
               writeln('Textul are si caractere speciale.');
               for i:=0 to 255 do
                   if chr(i) in b-a then write(chr(i),'  ');
               writeln
          end
end.
```

3.5 (V)

Să se stabilească proprietatea de palindrom a unui cuvânt. Un cuvânt este palindrom dacă, citindu-l de la stânga la dreapta sau de la dreapta la stânga se obţine acelaşi cuvânt. De exemplu: cojoc, sas – sunt palindroame.

Aceeaşi problemă pentru o frază dată. De exemplu: Ele fac cafele.

Scanned with OKEN Scanner

R 3.5

Vom considera întâi cazul unui cuvânt; el este palindrom dacă orice două caractere egal depărtate de extremităţile cuvântului sunt identice.

```
program palindrom;
var s:string;
    i,l:byte;
    pali:boolean;
begin
     write('Dati cuvantul: ');readln(s);
     pali:=true; l:=length(s);
     for i:=1 to l div 2 do
         if s[i]<>s[l-i+1] then pali:=false;
     if pali then writeln('Cuvantul este palindrom.')
             else writeln('Cuvantul nu este palindrom.');
     readln
end.
```

Cazul frazei palindromice necesită un studiu mai atent în sensul ignorării spaţiilor şi al altor caractere ortografice cu rol de separatori sau delimitatori de cuvinte (virgulă, punct, punct şi virgulă, linioară de unire, apostrof, ghilimele, semnul întrebării şi semnul exclamării). Într-o primă variantă vom ignora toate caracterele care nu sunt litere. În plus, nu se va face distincţie între literele mari şi cele mici considerând fraza scrisă numai cu litere mari (programul face automat acest lucru, folosind funcţia *upcase*).

```
program fraza_palindrom;
var s:string;
    i,j:byte;
    pali:boolean;
begin
     write('Dati fraza:');readln(s);
     pali:=true;
     for i:=1 to length(s) do s[i]:=upcase(s[i]);
     i:=1; j:=length(s);
     repeat
           while (s[i]<'A')or(s[i]>'Z') do i:=i+1;
           while (s[j]<'A')or(s[j]>'Z') do j:=j-1;
           if s[i]<>s[j] then pali:=false;
           i:=i+1;j:=j-1
     until (j<=i) or not pali;
     if pali then writeln('Fraza este palindromica')
             else writeln('Fraza nu este palindromica')
end.
```

O altă modalitate de lucru ar fi fost să se elimine din şir caracterele neliterale (inclusiv spaţiile) verificând pe şirul rămas proprietatea de palindrom; deoarece eliminarea caracterelor din şir afectează valoarea întoarsă de funcţia *length*, iar limitele contorului din bucla for sunt evaluate o singură dată, la început, nu se recomandă ştergeri în FOR, deci vom face o înlocuire prealabilă a acestora cu spaţii:

```
const litere=['A'..'Z'];
.... for i:=1 to length(s) do s[i]:=upcase(s[i]);
     for i:=1 to length(s) do if not(s[i] in litere) then s[i]:=' ';
     while pos(' ',s)<>0 do delete(s,pos(' ',s),1);
.....
```


Scanned with OKEN Scanner

3.6 (V)

Dintr-un şir de cuvinte date să se aleagă şi să se tipărească perechile polindromice (o pereche polindromică este formată din două cuvinte astfel încât, citind unul din ele de la dreapta spre stânga se obţine celălalt cuvânt).

R 3.6

Se citesc cuvintele şi se formează toate perechile de cuvinte scriind numai pe acelea care formează pereche polindromică. Se estimează lungimea maximă a unui cuvânt la 30 de caractere şi numărul maxim de cuvinte la 200:

```
program perechi;
type cuvinte=array[1..200]of string[30];
var c:cuvinte; p:boolean;
    n,i,j,k,l:byte;
begin
     write('Numar cuvinte:');readln(n);
     for i:=1 to n do begin
         write('Cuvantul nr. ',i,': ');
         readln(c[i])
     end;
     for i:=1 to n-1 do
         for j:=i+1 to n do
             if length(c[i])=length(c[j]) then begin
                l:=length(c[i]);
                p:=true;
                for k:=1 to l do
                    if c[i][k]<>c[j][l-k+1] then p:=false;
                    {litera de ordin k din cuvantul c[i] trebuie
                     sa coincida cu litera simetrica din c[j]}
                if p then writeln(c[i],'<---->',c[j])
             end;
     readln
end.
```

3.7 (V)

Se citeşte un text format din cel mult 255 de caractere. Să se înlocuiască literele mici cu literele mari corespunzătoare şi să se tipărească textul astfel obţinut. Aceeaşi problemă în care se cere înlocuirea tuturor literelor mari cu literele mici.

R 3.7

Se foloseşte funcţia *upcase(c:char)* care transformă literele mici în litere mari, lăsând neschimbate caracterele care nu sunt litere mici:

```
program lit_mari_mici;
var t:string; i:byte;
begin
     write('Dati textul:');
     readln(t);
     for i:=1 to length(t) do
         t[i]:=upcase(t[i]);     {*}
     write('Textul modificat:');
     writeln(t);
     readln
end.
```


Scanned with OKEN Scanner

Pentru conversia inversă se va înlocui linia {*} cu instrucţiunea:

```
if (t[i]>='A')and(t[i]<='Z') then t[i]:=chr(ord(t[i])+32)
```

unde 32 este diferenţa de cod ASCII între caracterul literă mică şi caracterul literă mare corespunzător

```
32=ord('a')-ord('A')=ord('b')-ord('B')=...
```

Dacă se lucrează cu o constantă de tip mulţime de caracter:

```
const litere=['A'..'Z'],
```

atunci instructiunea devine:

```
if t[i] in litere then t[i]:=chr(ord(t[i])+32)
```

Observaţie: În realitate, nu se pot citi de la tastatură mai mult de 127 de caractere (limita buffer-ului tastaturii); stringurile de lungime mai mare decât 127 se vor citi „pe bucăţi" sau dintr-un fişier text.

3.8 (VII)

Se citeşte un text de cel mult 125 de caractere format din cuvinte despărţite între ele prin exact câte un spaţiu. Să se afişeze numărul cuvintelor din text.

R 3.8

Se calculează numărul de cuvinte ca fiind cu 1 mai mare decât numărul de spaţii detectate în text. Evidenţierea spaţiilor se face cu ajutorul funcţiei *pos(subsir, sir:string)* care întoarce poziţia primei apariţii a unui subşir într-un şir şi a procedurii *delete(s,început,lung)* care şterge *lung* caractere din *s* începând de la poziţia *început*. Procedura *delete* este utilă pentru a elimina spaţiile deja detectate, în intenţia de a pune în evidenţă şi celelalte spaţii din text.

```
program cuvinte;
var t:string;
    cuv:byte;
begin
     write('Dati textul: ');readln(t);
     if t='' then cuv:=0 else cuv:=1;
     while pos(' ',t)<>0 do
           begin
                delete(t,pos(' ',t),1); cuv:=cuv+1
           end;
     write('Sunt in total ',cuv,' cuvinte');
     readln
end.
```

Acelaşi efect se obţinea cu structura de program:

```
if t='' then cuv:=0 else cuv:=1;
for i:=1 to length(t) do
    if t[i]=' ' then cuv:=cuv+1;
```

Dacă între cuvinte şi înaintea primului cuvânt sau după ultimul cuvânt se află un număr nelimitat (cel puţin 1) de spaţii şi/sau alţi separatori, atunci problema se complică necesitând numărarea cuvintelor alternativ cu trecerea peste separatori:

```
program cuvinte;
const car=['A'..'Z','a'..'z','0'..'9'];
var t:string;
    cuv,i:byte;
```


Scanned with OKEN Scanner

```
begin
      write('Dati textul: ');readln(t);
      cuv:=0; i:=1;
      t:=t+#10;           {marcaj de sfarsit de string}
      while t[i]<>#10 do begin
            while (t[i]<>#10) and not(t[i] in car) do i:=i+1;
            if t[i]<>#10 then cuv:=cuv+1;
            while t[i] in car do i:=i+1
      end;
      write('Sunt in total ',cuv,' cuvinte')
end.
```

Propunem ca exerciţiu căutarea altor algoritmi de rezolvare a problemei pentru acest caz.

3.9 (V)

Se introduce de la tastatură un text de cel mult 3 rânduri tastând ENTER o singură dată, la sfârşitul întregului text. Să se realizeze un program care permite înlocuirea repetată a unei anumite secvenţe cu altă secvenţă specificată.

R 3.9

Algoritmul prezentat – de eliminare a primei secvenţe s1 cu ajutorul procedurii *delete* şi de inserare pe aceeaşi poziţie a secvenţei *s2* – are un singur inconvenient. Urmăriţi programul:

```
program replace;
var t,s1,s2:string;
    p,i:byte;
begin
      writeln('Dati textul: ');
      readln(t);
      write('Dati secventa de inlocuit: ');
      readln(s1);
      while pos(s1,t)<>0 do
          begin
               write('Dati secventa cu care se inlocuieste: ');
               readln(s2);
               i:=0;
               while pos(s1,t)<>0 do
                     begin
                          i:=i+1;
                          p:=pos(s1,t);
                          delete(t,p,length(s1));
                          insert(s2,t,p)
                     end;
               writeln('S-au facut ',i,' inlocuiri')
          end
end.
```

Ce se va întâmpla dacă dorim înlocuirea secvenţei 'car' dintr-un text cu secvenţa 'carne'? Trebuie să evităm reluarea în discuţie a secvenţei din *t* în care s-au făcut deja înlocuirile lui *s1* cu *s2*. O metodă convenabilă este aceea prin care transferăm această secvenţă într-un alt şir:

Scanned with OKEN Scanner

```
program replace2;
var tt,t,s1,s2:string; {t=sirul sursa      tt=sirul destinatie}
   p,i:byte;
begin
     writeln('Dati textul: '); readln(t);
     write(' Dati secventa de inlocuit:'); readln(s1);
     while pos(s1,t)<>0 do begin
          write('Dati secventa cu care se inlocuieste:');readln(s2);
          tt:=''; i:=0;
          repeat
                i:=i+1; p:=pos(s1,t);
                tt:=tt+copy(t,1,p-1)+s2;
                {sau tt:=concat(tt,copy(t,1,p-1),s2)}
                delete(t,1,p-1+length(s1))
          until pos(s1,t)=0;
          t:=tt+t;{refacerea sirului sursa din sirul destinatie}
          writeln('S-au facut ',i,' inlocuiri')
     end
end.
```

3.10 (VI)

Se citeşte un text format din mai multe linii, după fiecare linie tastându-se ENTER, iar după ultima linie se tastează de două ori ENTER (o linie goală marchează faptul că s-a terminat de introdus întregul text). Să se stabilească procentual compoziţia textului: procent de vocale, consoane, cifre şi alte caractere.

Observaţii: – nu se va face deosebirea între literele mici şi cele mari;
– spaţiile vor fi ignorate (nu se vor considera caractere în text).

R 3.10

Se citesc, pe rând, liniile; parcurgând fiecare linie, numărăm fiecare fel de caractere ignorând spaţiile (le scădem din numărul total de caractere). Am folosit specificul de execuţie al instrucţiunii de selecţie CASE în PASCAL şi anume acela că se caută prima etichetă a cărei valoare coincide cu selectorul (caracterul t[*i*]), instrucţiunile următoare etichetate fiind ignorate; deci caracterele care sunt vocale vor duce la execuţia instrucţiunii *voc:=voc+1*, instrucţiunea *cons=cons+1* fiind ignorată, deşi sunt vocale şi-n lista de etichete 'A'..'Z'.

```
program compozitie;
var t:string;
    n_car,voc,cons,cif,alte,i:word;
begin
     n_car:=0;voc:=0;cons:=0;cif:=0;alte:=0;
     writeln('Introduceti pe rand liniile textului:'); readln(t);
     while t<>'' do begin
           n_car:=n_car+length(t);
           for i:=1 to length(t) do
               if t[i]=' ' then n_car:=n_car-1
               else case upcase(t[i]) of
                         'A','E','I','O','U':voc:=voc+1;
                         'A'..'Z':cons:=cons+1;
                         '0'..'9':cif:=cif+1
                          else alte:=alte+1
                        end;
            readln(t)
           end;
```


Scanned with OKEN Scanner

```
    if n_car=0 then writeln('Text vid')
    else begin
         writeln('Procent vocale:',voc*100/n_car:6:2);
         writeln('Procent consoane:',cons*100/n_car:6:2);
         writeln('Procent cifre:',cif*100/n_car:6:2);
         writeln('Procent alte caractere:',alte*100/n_car:6:2)
    end
end.
```

Observaţie: citirea fiecărei linii se poate realiza şi cu o structură de forma:

```
while not eoln do begin
     read (c); {c:char}
     ....
end; readln
```

3.11 (V)

Să se determine cele mai frecvente litere dintr-un text de cel mult 100 de caractere dat, tipărindu-se şi numărul aparţiilor lor.

R 3.11

Se memorează într-un tablou numărul de apariţii ale fiecărei litere, nefăcând deosebire între literele mari şi cele mici. Se determină elementul maxim al tabloului şi, prin parcurgerea acestuia încă o dată, se pun în evidenţă indicii (caracterele) poziţiilor egale cu maximul determinat.

Pentru a mări expresivitatea programului, am folosit o constantă tip mulţime ce reprezintă reuniunea (+) dintre mulţimea literelor mari şi mulţimea literelor mici. Procedura *inc(n)* este echivalentă pentru numere cu instrucţiunea *n:=n+1*.

```
program frecvente;
const litere=['A'..'Z']+['a'..'z'];
type frecv=array['A'..'Z'] of byte;
var f:frecv; t:string;
    i,max:byte; c:char;
begin
     write('Textul:');readln(t);
     for c:='A' to 'Z' do f[c]:=0;
     for i:=1 to length(t) do
         if t[i] in litere then inc(f[upcase(t[i])]); {*}
     max:=0;
     for c:='A' to 'Z' do
         if f[c]>max then max:=f[c];
     write('Cele mai frecvente litere sunt:');
     for c:='A' to 'Z' do
         if f[c]=max then write(c:2);
     writeln;
     writeln('Frcventa lor este ',max);
     readln
end.
```

Fără a folosi tipul mulţime, linia const litere = ['A'..'Z'] + ['a'..'z'] se elimină şi în locul liniei {*} se foloseşte instrucţiunea compusă

```
    begin
         t[i]:=upcase(t[i]);
         if (t[i]>='A')and(t[i]<='Z') then inc(f[t[i]])
    end;
```


Scanned with OKEN Scanner

3.12 (V)

Un ceas care sună la ore fixe poate fi potrivit să sune la oricare dintre cele 24 de ore din zi (numerotate de la 0 la 23). *N* persoane, fără să ştie una de cealaltă, potrivesc ceasul să sune la câte o oră. Dându-se orele la care a fost pus să sune ceasul (ore nu neapărat diferite), să se tipărească în ordine crescătoare orele la care va suna ceasul şi să se stabilească cel mai lung interval de linişte datorat ceasului astfel potrivit.

Exemplu: Se fac 7 reglaje la orele: 9,11,2,20,5,11 şi 22.

Se va tipări şirul: 2, 5, 9, 11, 20, 22 şi se va calcula intervalul maxim de linişte = 9 ore.

R 3.12

Se memorează într-o mulţime de numere din intervalul 0 ... 23 toate orele la care este fixat ceasul să sune. Dacă această mulţime nu este vidă, se vor stabili parcurgând tot intervalul orar al unei zile, acele ore la care va suna ceasul, memorând în variabilele *prima* şi *precedenta* prima oră şi respectiv ora precedentă la care este potrivit ceasul (pentru a calcula pe parcurs şi intervalul maxim de linişte).

```
program ceas;
var potriv:set of 0..23;
    n,i:byte;
    ora,prima,precedenta,max:0..23;
begin
     write('Numar de persoane care potrivesc ceasul: '); readln(n);
     if n=0 then begin
             write('Intreaga zi e linistita.'); halt
     end;
     potriv:=[];      {multimea vida}
     for i:=1 to n do begin
         repeat
               write('Persoana ',i,' potriveste ceasul la ora:');
               readln(ora);
          until ora in [0..23];
          potriv:=potriv+[ora]          {reuniune de multimi}
     end;
     for ora:=0 to 23 do
         if ora in potriv then begin
            prima:=ora;
            write('Ceasul va suna zilnic la orele:',prima,',');
            ora:=23 {iesire fortata din bucla FOR, echivalenta
                     cu procedura break in TURBO PASCAL 7.0}
         end;
     max:=0;
     precedenta:=prima;
     for ora:=succ(prima) to 23 do
         if ora in potriv then begin
            write(ora,',');
            if ora-precedenta>max then max:=ora-precedenta;
            precedenta:=ora
     end;
     if 24-precedenta+prima>max then max:=24-precedenta+prima;
     writeln(#8,' ');
     writeln('Intervalul maxim de liniste=',max,' ore'); readln
end.
```

Observaţie: Folosirea funcţiei *succ*(*x*) pentru *x* întreg este echivalentă cu expresia *x* + 1.

Scanned with OKEN Scanner

3.13 (VII)

Se citeşte de la tastatură – fără a afişa pe ecran – un cuvânt care va fi apoi ghicit prin încercări succesive de către o altă persoană. Cel care încearcă să afle cuvântul va propune o literă, iar calculatorul îi va răspunde tipărind cuvântul în care vor fi vizibile numai caracterele ghicite (celelalte sunt înlocuite cu '_'). Numărul de încercări este limitat la întregul cel mai apropiat de valoarea 20**dif*/*lung* (de exemplu), unde *dif* = numărul de caractere diferite din cuvântul propus, iar *lung* = numărul total de caractere ale acestuia.

R 3.13

Literele cuvântului vor fi memorate în mulţimea *lit*, iar literele ghicite în mulţimea *ghici*. Încercările sunt permise până ce mulţimile devin egale sau numărul de încercări depăşeşte valoarea *max* = *round(dif*20/lung)*:

```
program spanzuratoare;
uses crt;
var lit,ghici:set of char;
    cuv:string;l:char;
    lung,dif,max,incerc,i:byte;
begin
     textcolor(white);textbackground(black);clrscr; {alb pe negru}
     write('Cuvantul=');
     textcolor(black);readln(cuv);                  {negru pe negru}
     textcolor(white);                                {alb pe negru}
     lung:=length(cuv);dif:=lung;lit:=[];
     for i:=1 to lung do begin            {numara literele diferite}
         cuv[i]:=upcase(cuv[i]);
         if not (cuv[i] in ['A'..'Z']) then begin
            writeln('Cuvant eronat');halt
         end;
         if cuv[i] in lit then dec(dif);           {sau dif:=dif-1}
         lit:=lit+[cuv[i]]
     end;
     max:=round(20*dif/lung); ghici:=[];incerc:=0;
     while (ghici<>lit)and(incerc<max) do begin
           write('Ai ',max-incerc,' incercari sa ghicesti:');
           for i:=1 to lung do
               if cuv[i] in ghici then write(cuv[i]:2)
                                  else write('_':2);
           write(' Propune o litera:');
           repeat
                 l:=upcase(readkey)          {readkey=tasta apasata}
           until l in ['A'..'Z'];
           writeln(l); inc(incerc);
           if l in lit then ghici:=ghici+[l]
     end;
     if ghici=lit then begin
        sound(200);delay(250);nosound;
        writeln('Ai ghicit : ',cuv)
     end
     else begin
          sound(20);delay(250);nosound;
          writeln('Nu ai ghicit. Cuvantul era : ',cuv)
     end
end.
```


Scanned with OKEN Scanner

3.14 (VIII – C, Concurs ACM, faza zonală, variantă)

Se citeşte de la tastatură un text format din cel mult 120 de caractere (litere şi spaţii). Să se realizeze codificarea textului cu ajutorul alfabetului Morse. Se vor scrie codurile literelor, despărţite prin câte un spaţiu; între cuvinte se tipăresc 4 spaţii. Efectul sonor va fi o succesiune de sunete scurte (1/10 sec.) pentru '.' şi sunete mai lungi (3/10 sec.) pentru '–' despărţite între ele prin pauze scurte (1/10 sec.) între liniile şi punctele unui cod de literă, mai lungi (3/10 sec.) între litere, pauzele dintre cuvinte fiind cele mai lungi (1/2 sec).

Alfabetul MORSE este:

A .-	B -...	C -.-.	D -..	E .	F ..-.	G --.
H	I ..	J .---	K -.-	L .-..	M --	N -.
O ---	P .--.	Q --.-	R .-.	S ...	T -	U ..-
V ...-	W .--	X -..-	Y -.--	Z --..		

R 3.14

Se parcurge şirul caracter cu caracter transformând toate literele în majuscule. Pentru fiecare literă se scrie codul memorat în tabloul constant *cod* şi, în funcţie de caracterul următor (dacă este literă sau spaţiu) se scrie un singur spaţiu respectiv 4 spaţii. Caracterul ' ' adăugat la sfârşitul şirului (care nu se consideră din şir) are rolul figurativ de „caracter următor" pentru ultimul caracter din şir. Pentru controlul duratelor sunetelor s-a folosit în procedura *delay* (din unitatea CRT) o constantă (*cuanta*) reprezentând durata de 1/10 sec.

```
program cod_morse;
uses crt;
const cuanta=100;
      cod:array['A'..'Z'] of string[4]= ('.-', '-...', '-.-.',
      '-..', '.', '..-.', '--.', '....', '..', '.---', '-.-',
      '.-..', '--', '-.', '---', '.--.', '--.-', '.-.', '...',
      '-', '..-', '...-', '.--', '-..-', '-.--' , '--..');
var t:string; i,j:byte;
begin
     write('introduceti fraza: ');readln(t);
     t:=t+' ';
     for i:=1 to length(t)-1 do begin
         t[i]:=upcase(t[i]);
         if not(t[i] in ['A'..'Z',' ']) then
            begin writeln('Caracter nepermis');halt end;
         if t[i]<>' ' then begin
            write(cod[t[i]]);
            for j:=1 to length(cod[t[i]]) do
                if cod[t[i]][j]='.' then begin
                   sound(200);delay(cuanta);nosound;delay(cuanta)
                end
            else begin
                 sound(200);delay(3*cuanta);nosound;delay(cuanta)
            end;
            if t[i+1]=' ' then begin
               write('    ');delay(4*cuanta)
            end
            else begin write(' ');delay(2*cuanta) end
         end
     end
end.
```


Scanned with OKEN Scanner

3.15 (V – C)

Se citesc n nume de persoană ($1 \le n \le 40$). Să se afişeze aceste nume în ordine alfabetică de la dreapta la stânga pe n şiruri verticale despărţite prin coloane de caractere '|'.

Exemplu: pentru $n = 6$ numele:

```
Ion Dana
Mimi
Vlad Ion
Dorina
```

se vor afişa în forma:

```
D|I|M|V
o|o|i|l
r|n|m|a
i| |i|d
n|D| |
a|a| |I
 |n| |o
 |a| |n
```

R 3.15

S-a folosit o versiune de algoritm care scrie secvenţial câte un caracter din fiecare nume, intercalând caractere bara '|'. Spre deosebire de algoritmul care scrie numele unei persoane poziţionând fiecare caracter pe ecran cu procedura *gotoxy* (din unitatea CRT), programul prezentat va putea fi rulat şi pentru ieşire direcţionată la imprimantă.

```
program ordo_pers;
const lit_mari=['A'..'Z'];lit_mici=['a'..'z'];
      lit=lit_mari+lit_mici+['-',' '];
var pers:array[1..40]of string[25];n:byte; {Caracteristici persoane}
    max,i,j:byte; x:string; corect:boolean;  {Variabile de manevra}
begin
     repeat        {Introducera si validarea datelòr}
           write('Nr.persoane:'); readln(n)
     until n in [1..40];
     for i:=1 to n do
         repeat
               write('Persoana ',i,': '); readln(pers[i]);
               corect:=true;
               for j:=1 to length(pers[i]) do
                   if not(pers[i][j] in lit) then corect:=false;
               if not corect then write(#7)  {avertisment sonor}
         until corect;
     max:=length(pers[1]);                {numele de lungime maxima}
     for i:=2 to n do
         if length(pers[i])>max then max:=length(pers[i]);
     for j:=1 to n-1 do                        {ordonarea alfabetica}
         for i:=1 to n-j do
             if pers[i]>pers[i+1] then begin
                x:=pers[i]; pers[i]:=pers[i+1]; pers[i+1]:=x
             end;
     for i:=1 to max do begin                  {scrierea pe coloane}
         for j:=1 to n-1 do
             if i<=length(pers[j]) then write(pers[j][i],'|')
                                   else write(' |');
```


Scanned with OKEN Scanner

```
        if i<=length(pers[n]) then write(pers[n][i]);
        if i<max then writeln
    end
end.
```

3.16 (X)

Se dă un text (pe maxim 50 linii). Să se scrie textul pe două coloane de mărimi egale (de ziar). Lăţimea paginii este dată la intrare.

R 3.16

```
program pagina;
var line,text:array[1..50] of string;
    lintemp:string;
    nrlinii,lpag,lcol,numlin,linie,i,j,lc,cr:integer;
begin
    write('Dati numarul de linii de text: ');readln(nrlinii);
    writeln('Dati textul:');
    for i:=1 to nrlinii do readln(text[i]);
    write( 'Dati latimea paginii: '); readln(lpag);
    lcol:=lpag div 2-2;
    { se separa liniile in segmente de marimea lcol}
    lc:=1; cr:=1; lintemp := '';
    while cr<=nrlinii do begin
          lintemp:=lintemp+' '+text[cr];
          while lintemp[1]=' ' do delete(lintemp,1,1);
          while length(lintemp)>lcol do begin
                line[lc]:= copy(lintemp,1,lcol);
                while (line[lc][length(line[lc])]<>' ')
                      and(line[lc]<>'') do
                    delete(line[lc],length(line[lc]),1);
                if length(line[lc])=0 then begin
                   line[lc] := copy(lintemp,1,lcol);
                   delete(lintemp,1,length(line[lc]));
                end
                else begin
                          delete(lintemp,1,length(line[lc]));
                          delete(line[lc],length(line[lc]),1);
                     end;
                lc:=lc+1
          end;
          cr:=cr+1;
    end;
    if lintemp<> '' then line[lc]:=lintemp;
    linie:=lc;  {se aranjeaza liniile pentru a fi de lungimi egale}
     for i:=1 to linie do
         while length(line[i])<lcol do line[i]:=line[i]+' ';
     numlin:=linie div 2;    { se tiparesc liniile}
     if numlin*2<linie then numlin:=numlin+1;
     for i:=1 to numlin do begin
         for j:=1 to 2 do
             if (i+(numlin*(j-1))<=linie) then
                write(line[i+(numlin*(j-1))],' ');
         writeln
     end
end.
```


Scanned with OKEN Scanner

3.17 (IX – C, ACM, 1993)

Se consideră un fagure format din celule hexagonale realizate pe ecran cu ajutorul caracterelor '/','_','\', în unele dintre celule putându-se găsi regine reprezentate prin 'Q'. Să se reprezinte pe ecran fagurele ştiind că:

– sunt c coloane şi se cunoaşte pentru fiecare coloană numărul de celule de pe coloana respectivă;

– există r regine şi pentru fiecare regină se precizează coloana şi linia unde se găseşte aceasta;

– se ştie modul de decalare a celulelor de pe prima coloană faţă de a doua:

```
H(high) pentru  _          sau L (low) pentru    _
              / \_                               _/ \
              \_/ \                            / \_/
                \_/                            \_/
```

Exemplu:

c = 5, celule pe coloane: 3,5,2,1,4

r = 3, coordonate regine: (1,2) (1,3) (5,2)

tip decalaj: *H*

Desenul va trebui să arate în felul următor:

```
 _   _   _
/ \_/ \_/ \
\_/ \_/ \_/
/Q\_/ \_/Q\
\_/ \_/ \_/
/Q\_/   / \
\_/ \   \_/
  \_/   / \
  / \   \_/
  \_/
  / \
  \_/
```

R 3.17

De data aceasta am preferat o altă modalitate de lucru: se realizează în memorie fagurele ca tablou de stringuri în care o linie este unul dintre şirurile verticale ale desenului ($c1$ sau $c2$) şi se tipăreşte tabloul răsturnat (matricea transpusă), coloanele lui devenind linii. Tabloul reg memorează pentru fiecare linie şi coloană dacă în celula respectivă există sau nu regină.

```
program fagure;
const cifre=['0'..'9'];
    c0='                                                    ';
    c1='/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\';
    c2='_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ ';
type celule_vert=array[1..20]of 1..20;
    regine=array[1..20,1..20]of boolean;
    faguri=array[1..41]of string[42];
var cel:celule_vert;reg:regine;f:faguri;     {Caracteristici grafic}
    d:0..1;              {d=1--->deplasament L  d=0--->deplasament H}
    n1,n2,col,lin,i,j:byte;r:integer;
    ch:char;       {Variabile de manevra}
```


Scanned with OKEN Scanner

```
begin
     for i:=1 to 20 do
         for j:=1 to 20 do reg[i,j]:=false;
     for i:=1 to 41 do f[i]:=c0;
     lin:=0;
     write('Numar coloane=');readln(col);
     for i:=1 to col do
         begin
              write('Numar de celule pe coloana ',i,':');
              readln(cel[i]);
              if cel[i]>lin then lin:=cel[i]
         end;
     write('Numar regine=');readln(r);
     for i:=1 to r do
         repeat
              write('Coordonate regina ',i,':'); readln(n1,n2);
              reg[n2,n1]:=true
         until (n1>=1)and(n1<=col)and(n2>=1)and(n2<=cel[n1]);
     write('Decalajul primelor celule (L/H) :');
     readln(ch);
     d:=byte(upcase(ch)='L');
     {Realizarea tabloului grafic transpus}
     for j:=1 to col do
         begin
              f[2*j-1]:=copy(f[2*j-1],1,1+d)+copy(c1,1,2*cel[j]);
              f[2*j]:=copy(f[2*j],1,d)+copy(c2,1,2*cel[j]+1);
              f[2*j+1]:=copy(f[2*j+1],1,1+d)+copy(c1,2,2*cel[j]);
              for i:=1 to cel[j] do if reg[i,j] then
                  f[2*j,2*i+d]:='Q';
              d:=1-d;
         end;
     for i:=1 to 2*lin+2 do
         begin
              for j:=1 to 2*col+1 do write(f[j,i]);
              writeln
         end
end.
```

3.18 (IX – C, ACM, etapa zonală, 1994; Concurs PACO, 1995)

Se citeşte un şir de cel mult 70 de caractere reprezentând un număr natural foarte mare. Să se determine partea întreagă a rădăcinii sale pătrate şi diferenţa dintre numărul dat şi pătratul rădăcinii determinate.

R.18

Algoritmul calculului rădăcinii pătrate prin formarea grupelor de câte 2 cifre, coborârea grupelor una câte una, dublarea valorii rezultate, căutarea unei cifre care ...(etc.) este cunoscut de la matematică, din clasa a VI-a. Acelaşi algoritm este implementat în programul următor realizat de Vlad Marius din Alexandria:

```
program radical;
var n:string;
     i,j,i1,j1,k1,i2,k2,i3:longint;
     j2,cod:integer;
     q,q1,q2:string;
```


Scanned with OKEN Scanner

```
begin
      write('Numar: '); readln(n);
      if length(n) mod 2<>0 then n:='0'+n;
      val(copy(n,1,2),i,cod);
      j:=round(sqrt(i)); str(j,q);
      i1:=i-j*j;
      while length(n)>2 do begin
            str(i1,q1); delete(n,1,2);
            q1:=q1+copy(n,1,2); val(q,i3,cod);
            str(2*i3,q2); val(q1,i1,cod);
            j1:=0;
            for k1:=0 to 9 do begin
                q2:=q2+chr(48+k1); val(q2,i2,j2);
                if i2*k1<=i1 then begin
                   j1:=k1; k2:=i1-i2*k1
                end;
                delete(q2,length(q2),1)
            end;
            q:=q+chr(48+j1); i1:=k2
      end;
      str(i1,q1);
      writeln('Radacina patrata : ',q);
      writeln('Diferenta:',q1)
end.
```

3.19 (IX):

Ştiind prima zi a unui an, să se scrie în ce zi a săptămânii cade o anumită dată.

R 3.19

Răspunsul la problemă este foarte simplu: se determină numărul de zile scurs de la 1 ianuarie al anului respectiv, se află restul împărţirii la 7 şi se adună cu numărul asociat zilei de 1 ianuarie. Vom considera anul bisect.

```
program spt;
type luna=(ian,feb,mar,apr,mai,iun,iul,aug,sep,oct,nov,dec);
const sapt:array [1..7] of string=('luni','marti','miercuri',
              'joi','vineri','simbata','duminica');
      z:array[0..11] of
integer=(31,29,31,30,31,30,31,31,30,31,30,31);
      inceput:integer=0; {codifica ziua in care cade 1 ianuarie}
var b,c:luna ; x,y : integer;
begin
     writeln('Doriti sa stiti in ce zi cade o anumita data:');
     write('Ziua : ');read(y);
     write('Luna (numarul) :');readln(x);
     write('Presupunem ca 1 ianuarie a fost luni; atunci ');
     c:=ian;
     while ord(c)+1<x do begin
           y:=y+z[ord(c)];
           c:=succ(c);
     end;
     y:=((y mod 7)+inceput) mod 7 ;
     writeln('aceasta zi  este ',sapt[y])
end.
```

Observaţie: O variantă foarte frumoasă a acestei probleme este problema rezolvată 5.13.

Scanned with OKEN Scanner

3.20 (IX – C)

Într-un grup de n persoane se cunosc perechile (x, y) reprezentând numerele de ordine a două persoane cu proprietatea că orice noutate aflată de x este transmisă imediat lui y. Să se stabilească dacă în acest grup se va transmite un mesaj tuturor membrilor grupului, comunicându-i-l unei persoane date din grup.

R 3.20

Memorarea faptului că i îi transmite lui j mesajul se face atribuindu-i elementului din tabloul a de pe linia i şi coloana j valoarea 1 ($a[i, j] = 1$). Perechile i, j se citesc până la introducerea unei perechi incorecte (0, 0 – de exemplu). Mulţimea v conţine iniţial toate numerele de ordine ale persoanelor afară de una (prima – de exemplu), ea reprezentând grupul persoanelor neinformate. Deîndată ce o persoană j neinformată este anunţată de o persoană i informată ($a[i, j] = 1$,deci i îl anunţă pe j); j dispare din mulţimea persoanelor neinformate. În final, mulţimea v a persoanelor neinformate nu mai trebuie să conţină niciun element. Se foloseşte observaţia că transmiterea se face în cel mult n timpi.

```
program mesaj;
type multime=set of byte;
var a:array[1..20,1..20] of byte;
    n,i,j,k:byte;
    v:multime;
begin
     write('Cate persoane sunt in grup ? ');
     readln(n);
     v:=[2..n];
     fillchar(a,sizeof(a),0); {initializarea tabloului a cu zerouri}
     writeln('Scrieti perechile x y prin care circula mesajul:');
     readln(i,j);
     while (i in [1..n])and(j in [1..n]) do
           begin
                a[i,j]:=1;
                readln(i,j)
           end;
     k:=0;
     repeat
          for i:=2 to n do
              for j:=1 to n do
                  if not(i in v)and(j in v)and(a[i,j]=1) then
                     v:=v-[j];
          inc(k)
     until (k>n) or (v=[]);
     if v<>[] then
        writeln('Nu ')
     else
        writeln('Da');
     readln
end.
```


Scanned with OKEN Scanner

Probleme propuse

3.21 (VI)

Să se scrie toate sufixele unui şir de caractere. Se cer cel puţin doi algoritmi distincţi.

Observaţie: Un sufix este un şir rămas prin ştergerea unei secvenţe de caractere de lungime arbitrară (de la 0 la *length(s)* – 1) de la începutul şirului *s*.

3.22 (V)

Se citeşte un şir de caractere de lungime maximă 50. Să se afişeze sub formă de triunghi toate secvenţele mediane ale cuvântului, pornind de la întregul cuvânt, până la secvenţa alcătuită din caracterul (sau cele 2 caractere) din mijloc.

Exemplu: pentru şirul *Hipopotam* se afişează

```
Hipopotam
 ipopota
  popot
   opo
    p
```

Indicaţie: vezi şi problema 1.13

3.23 (V)

Se dă la intrare un număr n ($1 \le n \le 25$). Să se traseze un triunghi dreptunghic isocel cu catetele egale cu n, ca în exemplul de mai jos.

Intrare:

```
5
```

Ieşire:

```
* * * * *
*     *
*   *
* *
*
```

3.24 (VI)

Se citesc mai multe cuvinte. Să se listeze cuvintele în ordine alfabetică, cu litere mari.

De exemplu: pentru intrarea

```
aurel
var
curea
```

ieşirea va fi

```
AUREL
CUREA
VAR
```

3.25 (VI)

Se citesc un cuvânt şi o literă. Să se afişeze toate cuvintele obţinute prin înlocuirea fiecărui caracter al cuvântului cu litera dată.

Scanned with OKEN Scanner

3.26 (VI)

Un şir de caractere introduse de la tastatură trebuie să reprezinte numele şi prenumele unei persoane scris cu iniţiale majuscule, celelalte caractere fiind litere mici. Stabiliţi dacă şirul dat este un nume corect de persoană.

3.27 (VIII)

Să se determine, dintre n cuvinte date, perechile de cuvinte care rimează (au cel puţin două litere de la sfârşit identice).

3.28 (VII)

Pentru un text dat, se cere să se afişeze toate literele (fără a face deosebire între literele mari şi mici) folosite în acel text împreună cu frecvenţa de apariţie a fiecăreia.

3.29 (IX)

O expresie aritmetică simplă este formată din operanzi ce pot fi variabile notate cu o singură literă mică şi constante numere naturale, iar operatorii pot fi caractere din mulţimea {+,-,*,/}, fără paranteze. Să se stabilească dacă un şir dat este o expresie aritmetică simplă corectă.

3.30 (VIII – C, Olimpiada faza judeţeană, Vâlcea, 1995)

Se citeşte un cuvânt.

a) Să se determine codificarea „păsărească" a cuvântului, codificare ce constă din adăugarea după fiecare vocală din cuvânt a grupului format din caracterul **p** urmat de vocala respectivă.

b) Pentru un cuvânt codificat, să se determine cuvântul original.

3.31 (VIII – C, Concurs Eforie, 1994)

Se dă o listă cu n nume de persoane ($1 \leq n \leq 50$). Să se listeze toate numele de fete; dacă un nume apare de mai multe ori, se va scrie o singură dată.

3.32 (VII)

Se dau două texte. Să se stabilească o vocală comună ambelor texte, care apare de cele mai puţine ori.

3.33 (VI)

Se citeşte un text format numai din litere şi spaţii. Textul este introdus pe mai multe linii, sfârşitul acestuia fiind marcat de introducerea unei linii albe. Se citeşte apoi lăţimea efectivă a paginii exprimată în caractere.

Să se afişeze textul pe ecran, cu cuvintele spaţiate astfel încât toate liniile afară de ultima să fie complete (de la o margine la cealaltă a rândului).

Observaţie: Efectul de aliniere enunţat mai sus (justify) este utilizat în orice editor evoluat.

Scanned with OKEN Scanner

Exemplu: se citesc liniile textului:

```
  Iepurele, care era mai iute de picior,
o lua la sanatoasa si nu se opri pana ce
nu mai zari in urma sa pe arici.
```

Lăţimea paginii: 30.
Se va afişa textul aliniat astfel:

```
  Iepurele,  care era mai iute
de picior,  o lua la sanatoasa
si  nu  se opri pana ce nu mai
zari in urma sa pe arici.
```

3.34 (VII)

Pentru un număr dat se cere să se afişeze cifrele acestuia pe ecran, fiecare cifră fiind reprezentată cu ajutorul a 64 de caractere (8x8), caracterul '*' constituind „cerneala" şi caracterul '.' „fondul" de reprezentare a cifrei.

Comentariu:

a) Pentru fiecare cifră zecimală se defineşte un tablou de 8 şiruri de câte 8 caractere '*' şi '.' astfel încât, scriind fără spaţii liniile tabloului, acesta ar reprezenta chiar cifra respectivă. De exemplu pentru cifra 2 , tabloul

```
........
..****..
.*....*.
......*.
....**..
..**....
.******.
........
```

„desenează" chiar cifra 2.

b) Numărul se citeşte ca şir de caractere şi se validează folosind procedura val cu testarea celui de-al treilea parametru al acesteia. Se defineşte un tablou constant de 8 linii corespunzând liniilor tuturor cifrelor, şi 10 coloane corespunzând celor 10 cifre, componenta de rang i, j fiind stringul de 8 caractere ce formează linia i a cifrei j.

```
const c=array[1..8,'0'..'9']of string[8]=
(('........','........','........',etc.),
 ('..****..','....**..','..****..',etc.),
 ('.*....*.','...***..','.*....*.',etc.),
 ('.*....*.','..*..*..','......*.',etc.),
 ('.*....*.','.*...*..','....**..',etc.),
 ('.*....*.','.....*..','..**....',etc.),
 ('..****..','.....*..','.******.',etc.),
 ('........','........','........',etc.));
```

Scrierea şirului numeric *n* se va reduce la:

```
for i:=1 to 8 do
    begin
      for j:=1 to length(n) do write(c[i,n[j]]);
                    writeln
    end;
```


Scanned with OKEN Scanner

IV. STRUCTURI DE TIP ÎNREGISTRARE

Probleme rezolvate

4.1 (VI)

Se citesc data naşterii unei persoane şi data curentă. Să se determine vârsta în ani împliniţi a persoanei respective.

Exemplu:

```
data nasterii (zi,luna,an)= 12 7 1949
data curenta  (zi,luna,an)=  1 6 1996
```

Se afişează:

```
varsta= 46 ani împliniti.
```

R 4.1

Cele două date se pot memora fie ca două triplete (*a* [1], *a* [2], *a* [3]) şi (*n* [1], *n* [2], *n* [3]), fie ca trei perechi (*zi* [1], *zi* [2]), (*lun* [1], *lun* [2]) şi (*an* [1], *an* [2]) sau pur şi simplu prin şase numere (*zi_nast, lu_nast, an_nast, zi_crt, lu_crt, an_crt*). Algoritmul de prelucrare este acelaşi, structurarea datelor însă face ca programul să poată fi adaptat şi altor prelucrări sau nu (de exemplu într-o prelucrare complexă acesta poate fi folosit ca un subprogram care determină vârsta unei persoane în vederea determinării tuturor persoanelor cu vârsta între două limite date).

Pentru claritatea şi expresivitatea programului, dar mai ales în perspectiva lucrului cu informaţii memorate pe disc – fişiere –, recomandăm organizarea datelor sub formă de structuri neomogene numite înregistrări:

```
program ani_impliniti;
var n,c: record
                zi:1..31;
                lu:1..12;
                an:integer
         end;
    varsta:integer;
begin
     repeat
           write('Data nasterii (zi luna an) :');
           readln(n.zi,n.lu,n.an)
     until (n.zi in [1..31]) and (n.lu in [1..12]) and (n.an>0);
     repeat
           write('Data curenta (zi luna an) :');
           readln(c.zi,c.lu,c.an)
     until (c.zi in [1..31]) and (c.lu in [1..12]) and (c.an>0);
     varsta:=c.an-n.an;
     if (c.lu<n.lu)or ((c.lu=n.lu) and (c.zi<n.zi)) then
        varsta:=varsta-1;
     if varsta<0 then writeln('Date incorecte')
                 else writeln('varsta= ',varsta,' ani impliniti');
     readln
end.
```


Scanned with OKEN Scanner

4.2 (VI)

În registrul unei discoteci sunt trecute persoanele care intră şi vârstele lor (în ani şi luni). Ştiind că ultima persoană intrată este Dorina Vlad, să se calculeze media de vârstă a celor ce frecventează discoteca (în ani şi luni, aproximate la numere naturale).

R 4.2

Problema sugerează memorarea datelor persoanelor în tablouri (un tablou pentru nume şi două pentru vârste – sau un tablou cu două coloane sau mai simplu, un singur tablou cu vârsta calculată în luni). Observând însă că nu avem nevoie de aceste date în vederea prelucrărilor ulterioare, putem lucra cu datele referitoare la o singură persoană, structurate fie într-o înregistrare fie ca 3 date elementare (nume, ani, luni), având însă grijă să actualizăm suma vârstelor pentru fiecare „persoană intrată" şi să ţinem evidenţa numărului de persoane intrate. Analiza numelui fiecărei persoane este utilă pentru a sesiza sfârşitul şirului de date de intrare:

```
program discoteca;
const max=500;
      ulti1='DORINAVLAD';
      ulti2='VLADDORINA';
type persoana = record
                    nume:string[30];
                    ani,luni:byte
                end;
var p:persoana; ultima:boolean;
    med,i:word; nr:0..max;
begin
     nr:=0; med:=0;
     repeat
           nr:=nr+1;
           write('Numele :');readln(p.nume);
           repeat
                 write('Varsta (ani luni):');
                 readln(p.ani, p.luni)
           until p.luni in [0..11];
           med:=med+p.ani*12+p.luni;
           while pos(' ',p.nume)<>0 do
                 delete(p.nume,pos(' ',p.nume),1);
           for i:=1 to length(p.nume) do
               p.nume[i]:=upcase(p.nume[i])
     until (p.nume=ulti1)or(p.nume=ulti2);
     med:=med div nr;
   write('Varsta medie= ',med div 12,' ani si ', med mod 12,' luni')
end.
```

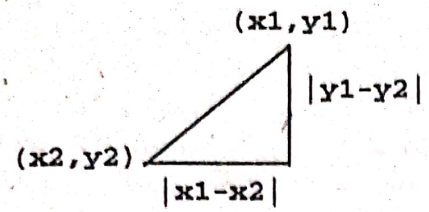
4.3 (IX)

Se citesc de la tastatură n perechi de numere reale reprezentând puncte din plan (în ordinea abscisă, ordonată). Scrieţi un program care să determine măsura celui mai mare segment având ca extremităţi două dintre punctele date.

R 4.3

Algoritmul de realizare a perechilor de puncte este prezentat şi în problema 3.6, iar calculul măsurii fiecărui segment este o aplicaţie a formulei lui Pitagora:

Scanned with OKEN Scanner



$$d(x_1, x_2) = \sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2}$$

```
program segmente;
type punct=record x,y:real end;
var p:array[1..500] of punct;
    n,i,j:integer;
    max,mas:real;  mai_este:boolean;
begin
     write('Numarul de puncte:');readln(n);
     for i:=1 to n do
         repeat
               write('Coord. punctului ',i,' (abscisa ordonata):');
               readln(p[i].x, p[i].y); mai_este:=false;
               for j:=1 to i-1 do
                   if (p[j].x=p[i].x)and(p[j].y=p[i].y) then
                      mai_este:=true;
               if mai_este then writeln('Ati mai dat acest punct')
         until not mai_este;
     max:=0;
     for i:=1 to n do
         for j:=i+1 to n do
             begin
                  mas:=sqrt(sqr(p[i].x-p[j].x)+sqr(p[i].y-p[j].y));
                  if mas>max then max:=mas
             end;
     writeln('Segmentul de lungime maxima are masura :',max:10:3)
end.
```

4.4 (VI)

Se introduc de la tastatură mediile generale, numele şi sexul elevilor dintr-o clasă; să se afişeze clasamentul la învăţătură al elevilor şi mediile lor în ordine descrescătoare, marcând cu o steluţă fetele şi cu un "#" băieţii cu medii până la 6 (exclusiv), cu două steluţe respectiv cu două caractere "#" elevii cu medii între 6 şi 9 (inclusiv) şi cu trei steluţe respectiv trei caractere "#" pe cei cu medii peste 9.

Exemplu: pentru 5 elevi

```
Marin Elena F 7.33
Ion Catalin M 9.65
Vasile Dana F 9.88
Ganea Dorin M 5.66
Tudor Vlad  M 5.33
```

se afişează:

```
Vasile Dana F 9.88***
Ion Catalin M 9.65###
Marin Elena F 7.33**
Ganea Dorin M 5.66#
Tudor Vlad  M 5.33#
```

R 4.4

Punctul de interes în cazul acestei probleme este algoritmul de ordonare care presupune inversiuni ale mediilor, dar împreună cu ele trebuie interschimbate şi

Scanned with OKEN Scanner

numele elevilor precum şi sexul, pentru a păstra informaţia corespunzătoare fiecărui elev. Dacă memorăm datele în tablouri diferite (de nume, de medii, de sexe), interschimbările de elemente se vor realiza anevoios (cu 9 atribuiri la fiecare inversiune). Putem structura toate datele referitoare la un elev într-o înregistrare, limbajul PASCAL permiţând atribuiri de date structurate cu condiţia ca acestea să fie de acelaşi tip (tipul *elev* declarat în instrucţiunea TYPE):

```
program clasament;
const nr_max=50;
type elev=record
                nume:string[30];
                medie:real;
                sex:char    {'B' pentru baieti si 'F' pentru fete}
          end;
var e:array[1..nr_max] of elev;
    aux:elev;              {inregistrare auxiliara de interschimbare}
    n,i,j,k:shortint;
    c:char;
    ordo:boolean;
begin
     write('Numarul de elevi=');readln(n);
     for i:=1 to n do
         begin
              writeln('Elevul ',i);
              write('Numele:');
              readln(e[i].nume);
              repeat
                    write('Baiat sau fata (B/F) ?');
                    readln(c);
                    e[i].sex:=upcase(c)
              until e[i].sex in ['B','F'];
              repeat
                    write('Media:');readln(e[i].medie)
              until (e[i].medie>=1)and(e[i].medie<=10)
         end;
     repeat
           ordo:=true;
           for i:=1 to n-1 do
               if e[i].medie<e[i+1].medie then
                  begin
                       aux:=e[i]; e[i]:=e[i+1]; e[i+1]:=aux;
                       ordo:=false
                  end;
     until ordo;
     writeln;writeln('CLASAMENTUL');
     for i:=1 to n do
         begin
           write(e[i].nume,'':30-length(e[i].nume),e[i].medie:7:2);
              if e[i].sex='B' then c:='#' else c:='*';
              if e[i].medie<6 then k:=1
              else if e[i].medie<=9 then k:=2
                   else k:=3;
              for j:=1 to k do write(c);
              writeln
         end
end.
```


Scanned with OKEN Scanner

Se recomandă, pentru a identifica direct câmpurile unei înregistrări, să se folosească instrucţiunea WITH. De exemplu secvenţa de scriere a rezultatelor devine:

```
for i:=1 to n do
    with e[i] do begin
         write(nume,'':30-length(nume),medie:7:2);
         if sex='B' then c:='#' else c:='*';
         k:=1+byte(medie>=6)+byte(medie>9)
         for j:=1 to k do write(c); writeln
    end
```

Observaţie: byte(medie>9)=1 dacă *medie>9*, operaţia *typecasting* care transformă valoarea unei variabile de un anumit tip în altă valoare de alt tip.

4.5 (X)

Cunoscându-se coeficienţii unui număr complex, să se afişeze modulul şi conjugatul său.

R 4.5

Suportul matematic (cunoaşterea noţiunilor şi formulelor) este aici determinant. Numărul complex este memorat folosind un tip de înregistrare cu 2 câmpuri.

```
program nr_complex;
type complex=record
                re,im:real
          end;
var n,nc:complex; modul:real;   {nc-numarul conjugat}
begin
    with n do begin
         writeln('Dati numarul complex:');
         write('Coeficientul partii reale:'); readln(re);
         write('Coeficientul partii imaginare:'); readln(im);
         nc:=n; nc.im:=-nc.im;
         modul:=sqrt(sqr(re)+sqr(im));
         writeln('Modulul este :',modul:10:3);
         write('Conjugatul este:',re:7:2);          {*}
         if im>0 then write('+');                   {*}
         writeln(im:7:2,'i')                        {*}
    end
end.
```

Pentru eliminarea spaţiilor şi zerourilor inutile din formatul de scriere se vor înlocui liniile {*} cu:

```
         if re=0 then restr:='' else str(re:7:2,restr);
         while restr[length(restr)] in ['0','.'] do
               delete(restr,length(restr),1);
         if im=0 then imstr:=''
         else begin
              str(-im:7:2,imstr);
              while imstr[length(imstr)] in ['0','.'] do
                    delete(imstr,length(imstr),1);
              imstr:=imstr+'i';
              if im<0 then imstr:='+'+imstr
         end;
         conj:=restr+imstr;
         while pos(' ',conj)<>0 do delete(conj,pos(' ',conj),1);
         writeln('Conjugatul=',conj)
```

declarând corespunzător `restr, imstr, conj:string.`

Scanned with OKEN Scanner

4.6 (VIII)

Pentru fiecare lună a anului se citeşte un şir de valori termice înregistrate în luna respectivă (numărul de valori nu depăşeşte 100, nefiind neapărat acelaşi pentru toate lunile). Să se afişeze alfabetic numele lunilor care au toate temperaturile înregistrate aflate între două valori termice date. Programul trebuie să permită introducerea repetată a celor două valori limită.

R 4.6

Se memorează 12 înregistrări, fiecare înregistrare conţinând informaţiile referitoare la o anumită lună (nume şi temperaturi); se ordonează alfabetic după nume, apoi se determină lunile cu valorile termice situate între două limite date.

```
const n_luni:array[1..12]of string[10]=('Ianuarie','Februarie',
              'Martie','Aprilie','Mai','Iunie','Iulie','August',
              'Septembrie','Octombrie','Noiembrie','Decembrie');
type luna=record
                  nume:string[10];
                  tinf,tsup:real
           end;
var l:array[1..12] of luna; aux:luna;
    i,j,k,nr:byte; cuv:string[4]; rasp:char;
    inf,sup,med:real;
begin
  for i:=1 to 12 do
     with l[i] do begin
        nume:=n_luni[i];writeln(nume);
        write('Nr.valori inregistrate:');readln(nr);
        write('Temperaturi inregistrate:');read(tinf);
        tsup:=tinf;
        for k:=2 to nr do begin
          read(med);
          if med>tsup then tsup:=med
          else if med<tinf then tinf:=med
        end;
        readln
     end;
     for i:=1 to 11 do
       for j:=i+1 to 12 do
         if l[i].nume>l[j].nume then begin
           aux:=l[i];l[i]:=l[j];l[j]:=aux
         end;
     cuv:='';
     repeat
       repeat
        write('Doriti sa ',cuv,'dati doua temp.(d/n)?');readln(rasp)
       until rasp in ['d','D','n','N'];
       if upcase(rasp)='D' then begin
         write('Temperatura 1:');readln(inf);
         write('Temperatura 2:');readln(sup);
         if inf>sup then begin med:=inf;inf:=sup;sup:=med end;
         for i:=1 to 12 do
           if (l[i].tinf>inf)and(l[i].tsup<sup)then
             writeln(l[i].nume)
       end;
       cuv:='mai '
     until upcase(rasp)='N'
end.
```


Scanned with OKEN Scanner

4.7 (X)

Pentru n ($n \leq 50$ dat) puncte din plan se cunosc: coordonatele carteziene (abscisă, ordonată) şi culoarea (o literă). Să se afle câte pătrate cu vârfurile de aceeaşi culoare se pot forma folosind punctele date.

R 4.7

Rezolvarea cea mai comodă constă în partiţionarea mulţimii de puncte în seturi de câte 4 puncte şi verificarea apoi a culorii şi relaţiilor dintre coordonate pentru a forma pătrate. Se obţine o accelerare a execuţiei (sensibilă numai pentru seturi foarte mari de date: $n > 2000$) dacă se consideră perechi de câte două puncte, deducând unde pot fi celelalte două astfel încât să formeze pătrat şi verificând dacă acestea sunt în mulţimea de puncte dată. Prezentăm aici prima dintre versiuni, lăsând ca exerciţiu scrierea programului pentru a doua variantă.

```
program patrate;
type tip=record
            x,y:real;
            cul:char
         end;
var a:array [1..30] of tip;
    s:array[1..6]of real;
    i,j,k,l,o:byte;  t,minc,maxc:integer;
    min,max:real;
begin
     write('Numar puncte:');readln(n);
     t:=0;
     for i:=1 to n do
         begin
              write('Coordonate (abscisa ordonata):');
              readln(a[i].x,a[i].y);
              write('Culoare:'); readln(a[i].cul)
         end;
     for i:=1 to n-3 do
         for j:=i+1 to n-2 do
             for k:=j+1 to n-1 do
                 for l:=k+1 to n do
                     if(a[j].cul=a[i].cul)and(a[k].cul=a[i].cul)
                         and(a[l].cul=a[i].cul) then begin
                        {4 laturi si 2 diagonale egale}
                        s[1]:=sqr(a[i].x-a[j].x)+sqr(a[i].y-a[j].y);
                        s[2]:=sqr(a[i].x-a[k].x)+sqr(a[i].y-a[k].y);
                        s[3]:=sqr(a[i].x-a[l].x)+sqr(a[i].y-a[l].y);
                        s[4]:=sqr(a[j].x-a[k].x)+sqr(a[j].y-a[k].y);
                        s[5]:=sqr(a[j].x-a[l].x)+sqr(a[j].y-a[l].y);
                        s[6]:=sqr(a[k].x-a[l].x)+sqr(a[k].y-a[l].y);
                        min:=9999;minc:=0;
                        for o:=1 to 6 do if min>s[o] then min:=s[o];
                        for o:=1 to 6 do if min=s[o] then inc(minc);
                        max:=0;maxc:=0;
                        for o:=1 to 6 do if max<s[o] then max:=s[o];
                        for o:=1 to 6 do if max=s[o] then inc(maxc);
                        if (minc=4) and(maxc=2)then inc(t)
                      end;
   writeln('Numarul de solutii este :',t)
end.
```


Scanned with OKEN Scanner

4.8 (VII)

La un depozit se lucrează cu cel mult 400 de sortimente de marfă. O factură de vânzare respectiv cumpărare conţine cel mult 10 tipuri de mărfuri vândute (cumpărate), pentru fiecare tip precizându-se cantitatea şi preţul. Ştiind că s-au înregistrat *m* astfel de facturi, să se precizeze:

– ce mărfuri au fost vândute fără a fi cumpărate (marfa din stoc);

– ce diferenţă bugetară există între vânzări şi cumpărări.

R 4.8

Aplicaţiile de gestiune folosesc date grupate în înregistrări, care sunt la rândul lor, organizate în fişiere. În acest caz, datele dintr-o factură sunt: tipul facturii (vânzare/cumpărare), număr de sortimente, pentru fiecare existând două valori numerice (cantitate, preţ unitar). Vom păstra o evidenţă a stocurilor de marfă pentru fiecare sortiment, adunând cantitatea de marfă intrată prin cumpărare şi scăzând cantitatea ieşită prin vânzare. Despre mărfurile cu stoc negativ putem spune că au fost vândute fără a fi cumpărate (fiind vândute dintr-un stoc existent anterior înregistrărilor din facturi):

```
type produs=record
                tip:1..400;
                cant:real;
                pret:word
          end;
     factura= record  {tipfact=1->cumparare; tipfact=-1->vanzare}
                    tipfact:shortint;
                    n_sort:1..10;
                    sort:array[1..10] of produs
               end;
var f:factura;  c:char;
    stoc:array[1..400] of real;
    nf,i,j:word; buget:real;
begin
     write('Nr.facturi:');readln(nf);
     buget:=0;
     for i:=1 to 400 do stoc[i]:=0;
     for i:=1 to nf do
         with f do begin
              writeln('Factura ',i,':');
              repeat
                    write('Tipul facturii (C/V):');
                    readln(c);c:=upcase(c)
              until c in ['V','C'];
              if c='C' then tipfact:=1 else tipfact:=-1;
              write('Nr. de sortimente:'); readln(n_sort);
              for j:=1 to n_sort do
                  with sort[j] do begin
                       write('Codul produsului ',j,':');readln(tip);
                       write('Cantitatea:');readln(cant);
                       write('Pret unitar:');readln(pret);
                       stoc[tip]:=stoc[tip]+cant*tipfact;
                       buget:=buget+cant*pret*tipfact
                  end
         end;
```


Scanned with OKEN Scanner

```
    write('Coduri produse vandute din stoc:');
    for i:=1 to 400 do
        if stoc[i]<0 then write(i:4);
    writeln;
    writeln('Balanta bugetara:', buget:20:3)
end.
```

4.9 (IX – C)

Se cunosc numele şi timpii obţinuţi la 5 probe de viteză (50m, 100m, 400m, 1200m şi 3000m) pentru n sportivi ($n \leq 100$). Să se formeze cât mai puţine grupe de sportivi şi să se afişeze numele lor, astfel încât:

– în nici o grupă să nu existe doi sportivi ocupanţi ai locului 1 la vreuna dintre probe;

– în toate grupele să fie acelaşi număr de sportivi.

R 4.9

Observaţia că sunt cinci astfel de grupe, este infirmată de un caz limită în care un singur sportiv este câştigătorul tuturor probelor, deci toţi sportivii pot fi puşi într-o singură grupă.

Rezolvarea problemei cuprinde două etape:

1. Aflarea numărului de grupe şi fixarea „liderilor" (câştigătorii probelor);
2. Distribuirea celorlaltor sportivi în grupe (dacă numărul lor se divide cu numărul de grupe determinat).

Nefiind nici o precizare în legătură cu modul de distribuire a celorlaltor sportivi, vom face o repartizare circulară (câte un sportiv în mulţimile 1, 2, ..., ng, 1, 2, .., ng, etc.). Deoarece sunt cel mult n grupe (cazul când există mai mulţi câştigători ai unei probe deci, la limită, toţi pot fi câştigători), vom lucra cu un tablou de mulţimi, în fiecare mulţime memorând numerele de ordine ale sportivilor dintr-o grupă.

```
program sportivi;
const dispo:set of byte=[1..100];  {sportivi nedistribuiti in grupe}
type grupa=set of 1..100;
     sportiv=record
                  nume:string[20];
                  t:array[1..5] of real;
             end;
var g:array[1..100] of grupa;
    s:array[1..100] of sportiv;
    ng,ns:byte;              {ng=numar grupe, ns=numar sportivi}
    rec:array[1..5] of real;   {timpii record la cele 5 probe}
    i,j:byte;
    lider:boolean;
begin
     write('Numar sportivi (cel mult 100):');readln(ns);
     for i:=1 to ns do                    {citirea datelor}
         with s[i] do begin
              writeln('Sportivul ',i);write('Numele:');readln(nume);
              for j:=1 to 5 do
                  repeat
                        write('Timp-proba ',j,':');readln(t[j])
                  until t[j]>0
         end;
```


Scanned with OKEN Scanner

```
    for j:=1 to 5 do begin              {stabilirea recordurilor}
        rec[j]:=s[1].t[j];
        for i:=1 to ns do
            if s[i].t[j]<rec[j] then rec[j]:=s[i].t[j]
    end;
    ng:=0;      {stabilirea numarului de grupe si a liderilor}
    for i:=1 to ns do begin
        lider:=false;
        for j:=1 to 5 do
            if s[i].t[j]=rec[j] then lider:=true;    {**}
            if lider then begin
               ng:=ng+1; g[ng]:=[i]; dispo:=dispo-[i]
            end
    end;
    if ns mod ng<>0 then writeln('Problema nu are solutie')
    else begin       {distribuirea sportivilor in grupe}
         j:=1;
         for i:=1 to ns do
             if i in dispo then begin
                g[j]:=g[j]+[i];
                j:=j*byte(j<ng)+1       {*}
             end;
         for j:=1 to ng do begin
             writeln('Grupa ',j,':');
             for i:=1 to ns do
                 if i in g[j] then writeln(s[i].nume);
             write('Apasati ENTER');readln
         end
    end
end.
```

Conform instrucţiunii {*}, j va primi valorile 1, 2, .., *ng*, 1, 2, .., *ng*, 1,.. etc.

Observaţie: Cu privire la instrucţiunea {**}, se recomandă, la testarea egalităţii a două numere reale, să nu se folosească relaţia propriu-zisă de egalitate (mai ales atunci când numerele se obţin din calcule), ci să se verifice dacă cele două numere diferă printr-o cantitate neglijabilă raportată la precizia de reprezentare a numerelor (se consideră deci $a = b$ dacă $abs(a - b) < 0.000001$).

4.10 (VI – C, Concurs ACM, 1993)

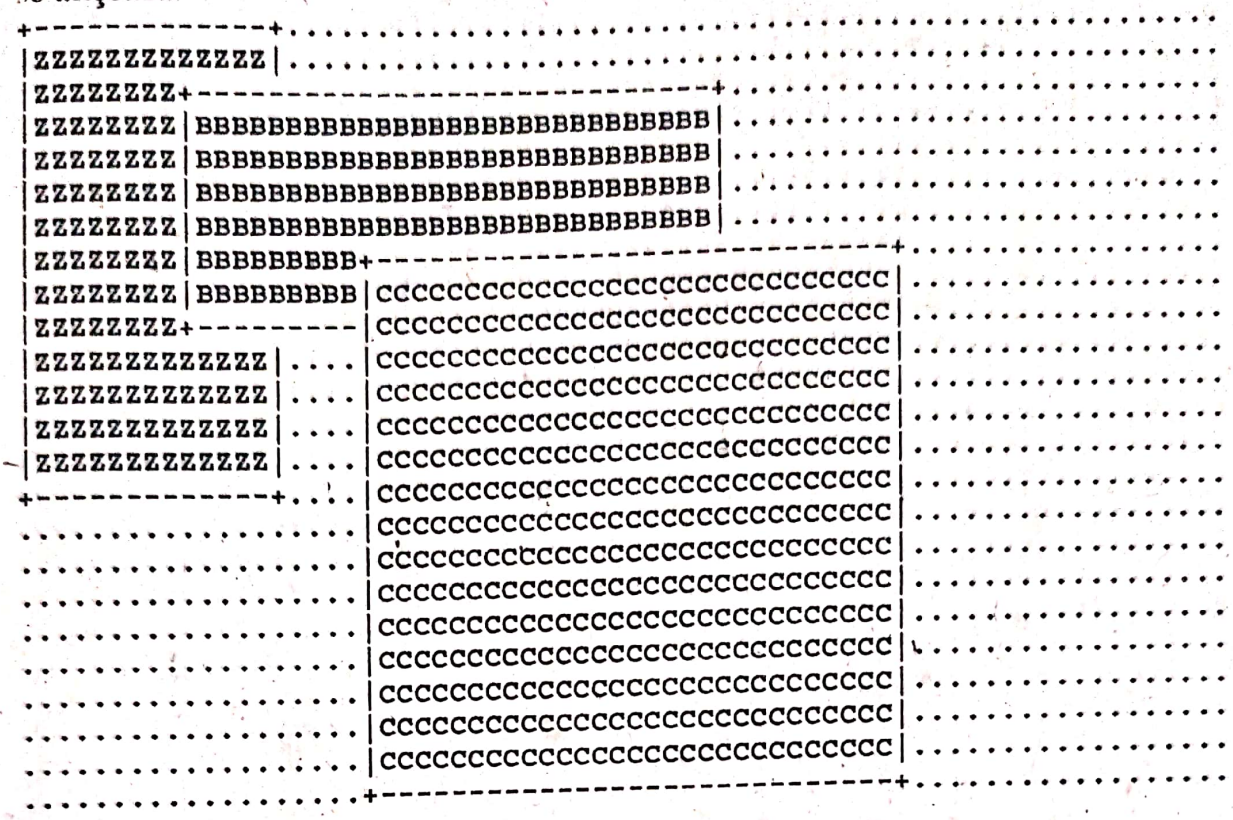
Se doreşte reprezentarea pe ecran a unor ferestre suprapuse. Pentru fiecare fereastră se citeşte o linie conţinând 6 numere: coordonatele (coloană linie) ale colţului din stânga-sus şi ale colţului din dreapta-jos, adâncimea ferestrei (fereastra de adâncime minimă va fi deasupra celorlalte, deci total vizibilă) şi codul ASCII al caracterului cu care se va umple fereastra. Se citesc de la dispozitivul de intrare mai multe linii până ce se dă o linie cu 6 valori nule. În reprezentarea marginilor, respectiv colţurilor ferestrelor se vor folosi caracterele '|','-' şi respectiv '+'.

Exemplu: pentru datele de intrare

```
10 3 40 10 0 66
20 8 50 24 -3 67
1 1 15 15 10 90
0 0 0 0 0 0
```


Scanned with OKEN Scanner

se afişează:



R 4.10.

```
program ferestre;
uses crt;
type punct=record
            x,y:byte;
          end;
     fereastra=record
                 car:char;
                 prof:real;          {profunzimea in stratificare}
                 ss,dj:punct
               end;
var desen:array[1..500] of fereastra;
    a,b,c,d,f:byte; e:real;  nf:word;
    i,j:word;aux:fereastra;gata:boolean;
    inter,marg,vid:string;
begin
     writeln('Dati cele 6 caracteristici ale unei ferestre:');
     readln(a,b,c,d,e,f);
     nf:=0;                                {numar ferestre}
     while a+b+c+d+abs(e)+f<>0 do          {test ultima linie}
           begin
                inc(nf);
                with desen[nf] do
                     begin
                          car:=chr(f);prof:=e;
                          ss.x:=a;ss.y:=b;dj.x:=c;dj.y:=d
                     end;
                writeln('Dati cele 6 caract. ale altei ferestre:');
                readln(a,b,c,d,e,f)
           end;
```


Scanned with OKEN Scanner

```
if nf=0 then halt;
repeat
      gata:=true;
      for i:=1 to nf-1 do
      if desen[i].prof<desen[i+1].prof then
         begin
              gata:=false;
              aux:=desen[i];
              desen[i]:=desen[i+1];
              desen[i+1]:=aux
         end;
until gata;
clrscr;              {asezarea ferestrelor pe ecran}
marg:='+';for i:=1 to 79 do marg:=marg+'-';
vid:='';for i:=1 to 80 do vid:=vid+'.';
for i:=1 to 24 do begin gotoxy(1,i);write(vid) end;
for i:=1 to nf do
    with desen[i] do
         begin
              inter:='|';
              for j:=2 to dj.x-ss.x do inter:=inter+car;
              gotoxy(ss.x,ss.y);
              write(copy(marg,1,dj.x-ss.x),'+');
              for j:=ss.y+1 to dj.y-1 do
                  begin
                       gotoxy(ss.x,j);
                       write(copy(inter,1,dj.x-ss.x),'|')
                  end;
              gotoxy(ss.x,dj.y);
              write(copy(marg,1,dj.x-ss.x),'+');
              gotoxy(1,25);
              write('Apasati o tasta pentru alta fereastra')
         end;
  writeln
end.
```

Observaţie: Pentru a prezenta o versiune cât mai scurtă de program, s-a considerat că datele de intrare sunt corecte. Erau potrivite însă nişte teste de validare a datelor (cu {$I-},{$I+} şi testarea funcţiei *ioresult* pentru a verifica dacă datele sunt numerice, cu compararea valorilor de vârf cu 1, 24, 80 pentru eventualele ferestre afară din ecran şi compararea valorii de cod ASCII cu 32 pentru eventualele caractere netipăribile). De obicei însă, în concursurile internaţionale de informatică se consideră apriori corectitudinea datelor de intrare.

4.11 (IX)

Se citesc de la tastatură datele referitoare la mai multe figuri geometrice codificate prin caracterele T (triunghi), C (cerc), D (dreptunghi) şi P (pătrat). Pentru cerc se cunoaşte raza, pentru pătrat latura, pentru triunghi laturile, iar pentru dreptunghi lungimea şi lăţimea. Să se tipărească toate datele figurilor cu arii ce diferă prin cel mult o unitate.

R 4.11

Pentru toate figurile se folosesc date de tip înregistrare cu variante, o înregistrare conţinând câmpurile comune *tip*, *arie* şi câmpurile specifice fiecărui tip de figură: *a*, *b*, *c* (laturile triunghiului), *r* (raza cercului), *l* (latura pătratului) şi

Scanned with OKEN Scanner

lung, *lat* (dimensiunile dreptunghiului). De remarcat că, deşi câmpul selector pentru alternative poate fi de orice tip, se preferă de cele mai multe ori tot un selector de tip scalar (numărabil) tocmai din cauză că selecţia de câmpuri din înregistrare se realizează în program printr-o instrucţiune CASE clasică.

```
type figura=record
                  aria:real;
                  case tip:char of
                  'C':(r:byte);
                  'P':(l:byte);
                  'D':(lung,lat:byte);
                  'T':(a,b,c:byte)
             end;
var n,i,j:byte; p:real;
    f:array[1..50] of figura;
begin
     write('Numar figuri:');readln(n);
     for i:=1 to n do
         with f[i] do begin
              repeat
                   write('Figura ',i,' este C/P/D/T:');
                   readln(tip); tip:=upcase(tip)
              until tip in ['C','P','D','T'];
              case tip  of
              'C': begin
                        write('Raza=');readln(r);
                        aria:=pi*sqr(r)
                   end;
              'P': begin
                        write('Latura=');readln(l);
                        aria:=sqr(l)
                   end;
              'D': begin
                        write('Dimensiunile=');
                        readln(lung,lat);
                        aria:=lung*lat
                   end;
              'T': begin
                        write('Laturile:');readln(a,b,c);
                        p:=(a+b+c)/2;
                        aria:=sqrt(p*(p-a)*(p-b)*(p-c))
                   end
               end
        end;
     for i:=1 to n-1 do
         for j:=i+1 to n do
             if abs(f[i].aria-f[j].aria)<1 then      {*}
               writeln('Fig.',i,' si ',j,' au arii aproape egale');
end.
```

Observaţii: Înregistrările cu variante sunt destul de des utilizate în programele care creează şi întreţin baze de date (este vorba de lucrul cu fişiere), însă programarea în limbajul PASCAL pentru aplicaţii de gestiune este oarecum demodată în contextul modernelor Sisteme de Gestiune a Bazelor de Date cum ar fi Paradox, FoxBase, FoxPro sau Acces.

În instrucţiunea IF marcată cu {*} trebuie să se realizeze, de fapt, o scriere mai complexă (prin care să se afişeze toate datele referitoare la cele două figuri), scriere pe care v-o lăsăm ca temă.

Scanned with OKEN Scanner

Probleme propuse

4.12 (X)

Se citeşte un string reprezentând un număr complex de forma $a + bi$ (de exemplu –3.6 + 4i) fără paranteze. Să se precizeze dacă forma introdusă este corectă.

4.13 (X)

Se citesc coeficienţii reali ai unor numere complexe (cel mult 40). Să se afişeze pe trei coloane numerele reale, numerele imaginare şi numerele cu componentele reale şi imaginare nenule din această listă.

Exemplu: pentru

```
3 -1
2  0
1  4
0 -1.5
- 2  0
```

ieşirea va fi:

```
2          -1.5i          3-i
-2                        1+4i
```

4.14 (X)

Se citesc coordonatele carteziene ale unor puncte din plan. Să se afişeze coordonatele polare corespunzătoare acelor puncte care se situează strict în interiorul cercului trigonometric de rază 10 centrat în origine.

4.15 (VI)

Se definesc n ($1 \le n \le 20$) ţări însoţite de culorile drapelelor lor. Se citeşte apoi de la tastatură o anumită culoare. Se cere listarea ţărilor care au acea culoare în componenţa drapelului.

4.16 (VIII)

Se dau n oraşe ($1 \le n \le 20$) împreună cu coordonatele lor (latitudine sud/nord, longitudine est/vest). Fiind nominalizat un oraş de la tastatură, să se determine cel mai apropiat oraş de acesta.

4.17 (IX)

Se dă un vector cu maxim n elemente ($1 \le n \le 100$). Fără a folosi tablouri suplimentare şi fără a modifica poziţia elementelor în tablou, să se listeze elementele din vector în ordine crescătoare, apoi descrescătoare.

Indicaţie: Fiecare element al vectorului este o înregistrare cu două câmpuri, unul este elementul propriu-zis, celălalt este indicele elementului în şirul ordonat (iniţial indicele elementului al i-lea este i).

4.18 (VII)

Se dau numele a n concurenţi şi notele obţinute la proba de gimnastică. Să se calculeze media notelor şi să se listeze concurenţii în ordinea clasării lor.

Scanned with OKEN Scanner

4.19 (IX)

Mărimile în radiani a n unghiuri ($n < 50$) se furnizează de la tastatură. Să se afişeze mărimile în grade minute şi secunde ale perechilor de unghiuri complementare dintre cele date.

Observaţii:

– în testarea condiţiei de complementaritate se va permite o toleranţă de cel mult 0.0001;

– reprezentarea mărimii în grade minute şi secunde se va face folosind numai numere întregi.

4.20 (VII)

Caracteristicile unui produs sunt date prin: preţul în lei la data intrării în magazin şi data până la care este garantată calitatea produsului. Ştiind că preţul scade progresiv cu aceeaşi sumă, astfel încât în ziua imediat următoare datei de expirare acesta să devină 0, să se calculeze preţul produsului la o anumită dată sau să se semnaleze că i-a expirat termenul de valabilitate.

4.21 (X)

Se citeşte un string reprezentând un număr complex de forma $z = a + bi$ (de exemplu –3.6 + 4i) fără paranteze. Coeficienţii a şi b nu pot fi simultan 0.

a) Să se precizeze dacă forma introdusă este corectă;

b) În caz afirmativ, se cere să se tipărească toate numerele complexe care au acelaşi modul şi fie aceeaşi parte reală, fie aceeaşi parte imaginară cu numărul dat.

4.22 (IX)

Un candidat la un concurs de admitere poate opta pentru unul dintre cele 3 profile de clase existente (A, B sau C) şi obţine o notă calculată ca medie a mai multor probe (număr real între 0 şi 10). El a optat pentru toate cele 3 profiluri într-o anumită ordine. Repartizarea candidaţilor se face în ordinea mediilor, la profilul la care au optat sau la profilele următoare, în ordine, dacă la cel curent nu mai sunt locuri. Cunoscându-se numărul de locuri aprobate pentru fiecare profil, numărul de candidaţi şi pentru fiecare candidat numele, opţiunile şi media, să se afişeze numele candidaţilor şi profilul la care au fost admişi.

4.23 (X)

Se citesc de la tastatură numele unor firme specializate în vânzarea componentelor electronice şi a echipamentelor de calcul. Fiecare firmă practică un adaos comercial p% cunoscut şi are o firmă furnizoare dintre firmele date anterior (furnizoarea primei dintre firme şi adaosul celei din urmă nu se introduc). Cunoscându-se preţul unui calculator vândut la una dintre firme, să se afişeze preţul aceluiaşi calculator vândut de altă firmă dată.

Exemplu: pentru 5 firme

```
ALICOMP     5%
MARA        20%     ALICOMP
DONA        8%      ALICOMP
ASTRO       10%     MARA
COMPUT              ASTRO
```

şi preţul de 3000000 lei la firma MARA, dacă ni se cere preţul de la firma ASTRO, se va afişa valoarea 3300000 lei.

Scanned with OKEN Scanner

V. PROCEDURI ŞI FUNCŢII DEFINITE

Probleme rezolvate

5.1 (V)

Se citesc vârstele în ani ale mai multor persoane. Încercaţi să le aşezaţi la mese astfel încât persoanele de la aceeaşi masă să aibă aceeaşi vârstă. Să se afişeze numărul de mese.

R 5.1

Problema presupune numărarea elementelor distincte dintr-un şir dat. Deoarece elementele sunt numere naturale din intervalul [0, 150], reducem problema la construirea mulţimii vârstelor respective şi apoi la calculul cardinalului ei. Procedura *date* tratează într-o „anexă" a programului segmentul de introducere a datelor. Funcţia *card* transmite programului principal numărul de elemente din mulţimea vârstelor (v). Atât procedura cât şi funcţia sunt apelate o singură dată în program şi lucrează cu variabilele globale ale programului, deci se putea renunţa la ele transferându-se instrucţiunile în programul principal care însă rămânea stufos şi greu de urmărit de către cineva neinteresat de toate amănuntele de programare:

```
program banchet;
const vmax=255;                          {varsta maxima}
type multime=set of 1..vmax;
var v:byte;                              {varsta data}
    m:multime;                           {multimea varstelor}
    n,nn,i:integer;
procedure date;                          {citirea varstelor}
begin
     write('numar persoane=');readln(n);
     m:=[];
     for i:=1 to n do
         begin
              write('Varsta persoanei nr.',i,'='); readln(v);
              m:=m+[v]
         end
end;

function card:byte; {calculeaza nr.de elemente ale multimii}
begin
     nn:=0;
     for i:=1 to nmax do if i in m then inc(nn);
     card:=nn
end;

begin
     date;
     writeln('Numar de mese necesar=',card);
     readln
end.
```


Scanned with OKEN Scanner

5.2 (V)

Se citesc de la tastatură n ($n \leq 20$) numere întregi foarte mari (de cel mult 999 de cifre). Să se afişeze suma lor.

R 5.2

Programul simulează adunarea a două numere cifră cu cifră pe doi vectori de lungime 1020. Deşi se putea lucra şi în altă manieră, am preferat-o pe aceasta pentru a evidenţia corespondenţa parametrilor formali cu cei actuali şi mecanismul transmiterii valorilor rezultat prin parametrii valoare declaraţi cu VAR:

```
program suma;
const lung=1020;
type numar=array[1..lung] of 0..9;
var t1,t2:numar; n,i:integer;
procedure citeste(var x:numar);
  var i,j:integer;c:char;
  begin       {numarul apare rasturnat in vector}
       while not eoln do begin
             read(c);
             if not(c in ['0'..'9']) then halt;
             for j:=lung downto 2 do x[j]:=x[j-1];
             x[1]:=ord(c)-48;
       end;
       readln
  end;
procedure aduna(var n1,n2:numar);
  var i,j,k:integer;
  begin        {de la ultima la prima cifra}
       k:=0;
       for i:=1 to lung do begin
           k:=k+n1[i]+n2[i];
           n1[i]:=k mod 10;
           k:=k div 10
       end
  end;
procedure scrie(var x:numar);
  var j:integer;
      b:boolean;
  begin    {scrierea cu ignorarea cifrelor nesemnificative din fata}
       b:=false;
       for j:=lung downto 1 do begin
           if x[j]>0 then b:=true;
           if b then write(x[j]);
       end;
       if not b then write(0); writeln
  end;
begin
     write('Cate numere doriti sa adunati ? ');
     readln(n);
     fillchar(t1,lung,0);
     write('Primul numar:');citeste(t1);
     for i:=2 to n do begin
         fillchar(t2,lung,0);
         write('Numarul ',i:2,':');citeste(t2);aduna(t1,t2)
     end;
     scrie(t1)
end.
```


Scanned with OKEN Scanner

5.3 (VI)

Pentru a spori precizia de calcul, un programator trebuie să calculeze suma a *n* fracţii de numere întregi, efectuând o singură împărţire reală. Cum procedează?

Observaţie: Dacă pentru calculul expresiei *a/b + c/d* cu *a, b, c, d* întregi, se efectuează împărţirea reală *a/b*, apoi *c/d*, eroarea de la fiecare calcul se cumulează.

R 5.3

Se adună la fracţia iniţială fiecare dintre fracţiile următoare prin metoda „aducerii la acelaşi numitor" făcând toate simplificările posibile. Pentru a putea urmări la fiecare pas corectitudinea de calcul se vor afişa rezultatele intermediare.

```
program fractii;
type fractie=record   {nmr=numarator, nmi=numitor}
                nmr,nmi:longint
             end;
var fdat,frez:fractie; n,i:byte;
function cmmdc(a,b:longint):longint;
var r:longint;
begin
     repeat
          r:=a mod b; a:=b; b:=r
     until r=0;
     cmmdc:=a
end;
procedure simplifica(var f:fractie);
var fact:longint;
begin
     fact:=cmmdc(f.nmr,f.nmi);
     f.nmr:=f.nmr div fact;  f.nmi:=f.nmi div fact
end;
procedure suma(f:fractie;var g:fractie); {g=f+g}
var divcom:longint;                    {divizor comun}
begin
     {numitorul comun=cmmmc(nmr,nmi)=nmr*nmi/cmmdc(nmr,nmi)}
     divcom:=cmmdc(f.nmi,g.nmi);
     g.nmr:=f.nmi div divcom*g.nmr+g.nmi div divcom*f.nmr;
     g.nmi:=f.nmi div divcom * g.nmi
end;
begin
     write('Cate fractii adunati? ');readln(n);
     repeat
          write('Prima fractie (numarator numitor):');
          readln(frez.nmr,frez.nmi)
     until frez.nmi<>0;
     simplifica(frez);
     for i:=2 to n do begin
         writeln('Rezultat partial ',frez.nmr,'/',frez.nmi);
         repeat
              write('Fractia ',i,' (numarator numitor):');
              readln(fdat.nmr,fdat.nmi)
         until fdat.nmi<>0;
         simplifica(fdat);
         suma(fdat,frez);
              simplifica(frez);
     end;
     with frez do writeln('Rezultat',nmr,'/',nmi,'=',nmr/nmi:20:10);
end.
```

Observaţie: Se preferă ca în şirul de operaţii *nmr*nmi div c* să se efectueze *nmr div c* nmi* sau *nmi div c* nmr*, deoarece înmulţirea *nmr*nmi* poate depăşi tipul *longint*.

Scanned with OKEN Scanner

5.4 (VIII)

Să se afişeze tabelul de semne al unei expresii raţionale de forma

$$\frac{(a_1x+b_1)(a_2x+b_2)\ldots(a_kx+b_k)}{(c_1x+d_1)(c_2x+d_2)\ldots(c_nx+b_n)}$$

cu $k, n \leq 10$ numere naturale şi a_i, b_i, c_i, d_i numere reale date.

R 5.4

Se memorează într-un vector t tăieturile realizate de soluţiile ecuaţiilor $ax + b = 0$, iar în vectorul s caracterele '+'/'–' în funcţie de semnul expresiilor pe intervalul ($t[i\text{-}1]$, $t[i]$). Versiunea de mai jos nu stabileşte rezultatul expresiei în punctele de tăietură (dacă este 0 sau nu are sens –infinit!–):

```
program tabel_semne;
const max=10;
type taieturi=array[1..2*max] of real;
     semne=array[0..2*max] of char;
var t:taieturi; s:semne; mesaj:string;
    a,b:real; i,j,k,l,n_tai:byte;
procedure insereaza_taietura(var j:byte);
var i:byte;   {insereaza valoarea -b/a in lista de taieturi}
begin
     j:=1;                          {cautare}
     while(-b/a>t[j]) and (j<=n_tai) do inc(j);
     if j<=n_tai then if -b/a=t[j] then exit;
     inc(n_tai);                    {inserare}
     for i:=n_tai downto j+1 do
         begin t[i]:=t[i-1];s[i]:=s[i-1] end;
     t[j]:=-b/a; s[j]:=s[j-1]
end;
procedure schimba(i,j:byte);        {schimba semnele}
var k:byte;                         {coduri ASCII:43(+) si 45(-)}
begin
     for k:=i to j do s[k]:=chr(43+2*byte(s[k]='+'))
end;
begin
     fillchar(s,sizeof(s),'+');
     write('Numar de factori la numarator:');readln(k);
     write('Numar de factori la numitor:');readln(l);
     n_tai:=0;
     for i:=1 to l+k do begin
         write('ax+b= (introduceti a si b cu spatiu):');readln(a,b);
         if a=0 then
            if b=0 then begin
               if i<=k then mesaj:=' este 0 peste tot.'
                       else mesaj:=' nu are sens.';
               writeln('Expresia ',mesaj); halt
            end
            else if b<0 then schimba(0,n_tai) else
         else begin
              insereaza_taietura(j);
              if a>0 then schimba(0,j-1) else schimba(j,n_tai)
         end
     end;
     write('      ');
```


Scanned with OKEN Scanner

```
     for i:=1 to n_tai do write(t[i]:6:2);
     writeln;
     for i:=0 to n_tai do write(s[i]:6);
     readln
end.
```

5.5 (IX)

Se consideră funcţia de două variabile reale:

$$f(x)=\begin{cases} x^2+y^2, \text{ dacă } \max(|x|,|y|)<1 \\ \frac{1}{x}+\frac{1}{y}, \text{ dacă } \min(|x|,|y|)>1 \\ x+y, \text{ altfel} \end{cases}.$$

Se citesc de la tastatură trei perechi de numere (x, y) şi se cere să se scrie cea mai mare şi cea mai mică dintre valorile corespunzătoare ale funcţiei.

Exemplu: pentru datele (4, 2) (0.25, 0.64) şi (1, 0.25) obţinem f(4, 2) = 0.75; f(0.25, 0.64) = 1.3; f(1, 0.25) = 1.25 şi se vor afişa valorile 1.25 şi 0.75.

R 5.5

Conceperea secvenţei de calcule pentru valoarea funcţiei ca subprogram de tip FUNCTION este justificată: în acest mod, programul devine mai „aerisit" şi mai scurt. Subprogramul se scrie o singură dată şi se apelează în trei împrejurări diferite; parametrii efectivi (*x*1, *y*1), (*x*2, *y*2), (*x*3, *y*3) vor înlocui parametrii formali (*x*, *y*) care au la definirea funcţiei rol de „figuranţi". Nu s-a făcut nici o structurare a datelor tocmai pentru a urmări doar mecanismul de apelări ale funcţiilor *f*, *min* şi *max*:

```
program functie;
var x1,y1,x2,y2,x3,y3,f1,f2,f3:real;
function max(a,b:real):real;
begin
     if b>a then max:=b else max:=a
end;
function min(a,b:real):real;
begin
     if b<a then min:=b else min:=a
end;
function f(x,y:real):real;
begin
    if max(abs(x),abs(y))<1 then f:=sqr(x)+sqr(y)
    else if min(abs(x),abs(y))>1 then f:=1/x+1/y
                                 else f:=x+y
end;
begin
     write('Dati prima pereche de valori:');
     readln(x1,y1); f1:=f(x1,y1);
     write('Dati a doua pereche de valori:');
     readln(x2,y2); f2:=f(x2,y2);
     write('Dati prima pereche de valori:');
     readln(x3,y3); f3:=f(x3,y3);
     writeln('Valoarea maxima=',max(f1,max(f2,f3)):10:3);
     writeln('Valoarea minima=',min(f1,min(f2,f3)):10:3)
end.
```


Scanned with OKEN Scanner

5.6 (V)

Să se elaboreze un program care transformă un număr natural dat (din intervalul [1, 1000]) în secvenţa literală corespunzătoare numărului respectiv scris în limba română. Se va avea în vedere o modalitate de a permite execuţia repetată a transformărilor.

Exemplu: pentru

```
23
```

se va scrie la ieşire:

```
douazeci si trei
```

R 5.6

Procedura *wri* scrie corespondentul literal al cifrei *c*, în funcţie şi de eventualul acord în gen care trebuie să fie făcut pentru 1 şi 2. Opţiunea de compilare {$I-} inhibă (anulează) întreruperea execuţiei programului în cazul detectării unei erori de intrare/ieşire, starea de eroare putând fi controlată prin program şi ieşirea propriu-zisă din starea de eroare fiind anulată numai prin apelul funcţiei sistem IORESULT. Neconsultarea acesteia va conduce la semnalarea erorii de intrare/ ieşire odată cu prima instrucţiune de scriere/citire care urmează celei care a generat eroarea. În cazul de faţă se doreşte controlarea situaţiei în care de la tastatură se furnizează un şir incorect din punct de vedere numeric pentru *n* (Eroarea "Invalid numeric format").

```
program nr_literal;
type gen=(fem,masc);
var n,s,z,u:word;
    b:gen;
procedure wri(c:byte;b:gen);
begin
     case c of
          1:if b=masc then write('unu')
                      else write('o');
          2:if b=masc then write('doi')
                      else write('doua');
          3:write('trei');
          4:write('patru');
          5:write('cinci');
          6:write('sase');
          7:write('sapte');
          8:write('opt');
          9:write('noua');
     end
end;

procedure citeste;
begin
     repeat
          write('Dati un nr. intre 1 si 1000 (0-pentru oprire):');
          {$I-} readln(n) {$I+}
     until (ioresut=0) and (n<=1000)
end;
```


Scanned with OKEN Scanner

```
begin
     citeste;
     while n<>0 do begin
          case n of
               100 :writeln('o suta');
               1000 :writeln('o mie');
               else begin
                    s:=n div 100;n:=n mod 100;
                    if s<>0 then begin
                       wri(s,fem);
                       if s=1 then write(' suta ')
                              else write(' sute ')
                    end;
                    z:=n div 10;u:=n mod 10;
                    if z<>0 then
                       if z=1 then write(' zece ')
                       else begin
                                 wri(z,fem);write(' zeci ')
                            end;
                    if u<>0 then begin
                       if z<>0 then write('si ');
                       wri(u,masc)
                    end
               end
          end;  {case}
          readln;
          citeste
     end
end.
```

Observaţii: Pe acelaşi principiu poate fi gândit şi algoritmul pentru numere mai mari; rămâne însă de rezolvat (pentru acest caz ca şi pentru cel propus) problema numerelor între 11 şi 19 la care vă lăsăm plăcerea să vă gândiţi singuri.

5.7 (XI)

Să se rezolve ecuaţia $2^x = x^2$ prin metoda înjumătăţirii intervalului, când se citesc valorile reale reprezentând capetele intervalului pe care se caută soluţia şi numărul n de zecimale cu care se doreşte afişarea soluţiei.

R 5.7

Se aduce ecuaţia la forma $f(x)=0$. Conform unei consecinţe a teoremei lui Darboux, dacă o funcţie continuă îşi schimbă semnul pe un interval $[a, b]$, atunci există un punct c în acest interval în care funcţia se anulează. Se studiază comportarea funcţiei în punctul de mijloc al intervalului – $jum=(a+b)/2$ – şi se păstrează în obiectiv subintervalul pe care funcţia îşi schimbă semnul – (a, jum) sau (jum, b) – continuând căutarea până ce mărimea intervalului de căutare devine mai mică decât 10^{-n}, când se poate concluziona că orice număr din acel interval este o bună aproximantă cu n zecimale a soluţiei sau când se cade pe un punct de jumătate care anulează funcţia. Pe intervalul pe care funcţia nu îşi schimbă semnul, fie nu există soluţii, fie există un număr par de soluţii şi se recomandă alegerea altui interval sau a altei modalităţi de rezolvare. (Pentru alte detalii consultaţi [8])

Scanned with OKEN Scanner

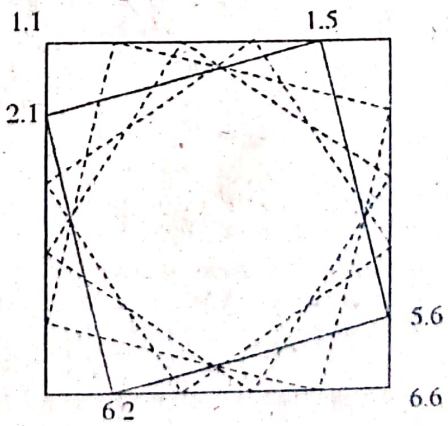
```
program bisectie;
var a,b,j,eps:real;
    n_zec,i:byte;
function e(x:real):real;
begin
    e:=exp(x*ln(2))-x*x
end;
begin
    writeln('Pe ce interval [a,b] cautati solutia?');
    write('a=');readln(a);write('b=');readln(b);
    write('Cate zecimale doriti ?');readln(n_zec);
    eps:=1;
    for i:=1 to n_zec+1 do eps:=eps/10;
    if(e(a)*e(b)<0)then begin
      repeat
            j:=(a+b)/2;
            if(e(j)*e(a)<0) then b:=j else a:=j
      until (abs(a-b)<eps)or(e(j)=0);
      writeln (j:30:n_zec)
    end
    else if abs(e(a))<eps then writeln(a:30:n_zec)
         else if abs(e(b))<eps then writeln(b:30:n_zec)
              else writeln('cautati alt interval');
end.
```

5.8 (VIII – C)

Într-o livadă sunt *m* şiruri de meri, pe fiecare şir aflându-se *n* pomi. Privită din avion, livada se aseamănă cu o tablă de GO în nodurile căreia sunt pomi. Se cunoaşte numărul de mere din fiecare pom (numărarea acestora fiind o sarcină a copiilor din sat). Primarul satului are dreptul să culeagă roadele a patru pomi cu condiţia ca ei să formeze un pătrat. Scrieţi un program care determină – pe baza datelor problemei – care sunt coordonatele celor 4 pomi (şirul şi al câtelea pom din şir este) astfel încât numărul merelor culese să fie maxim.

R 5.8

Pentru fiecare pom se cercetează toate pătratele care se pot forma cu acel pom în colţul din stânga sus. Sunt pătrate de forma:



Un pătrat este descris prin câmpurile înregistrării de tip *pătrat*, unde *i*, *j* sunt coordonatele vârfului din stânga sus, *lat* este latura pătratului în care se înscrie, iar *rot* este numărul de unităţi cu care este rotit acesta în pătratul (cu laturile paralele cu axele) în care se înscrie.

Scanned with OKEN Scanner

Exemplu: pătratul (1, 1, 5, 4) este pătratul cu colţurile (1, 5) – (5, 6) – (6, 2) – (2, 1) rotit în pătratul (1, 1) – (1, 6) – (6, 6) – (6, 1) cu patru unităţi.

```
program livada_de_meri;
type livada=array[1..100,1..10]of longint;
     patrat=record i,j,lat,rot:byte end;
var l:livada;
    m,n:byte;
    meremax,mere:longint;
    p,pmax:patrat;

procedure citeste;
........

procedure scrie;
........

function min(a,b:byte):byte;
begin
     if a>b then min:=b else min:=a
end;

function calc:longint;
begin            {calculeaza suma merelor din varfurile unui patrat}
     with p do
     calc:=l[i,j+rot] + l[i+rot,j+lat] +
           l[i+lat,j+lat-rot] + l[i+lat-rot,j]
end;
begin
     citeste;
     meremax:=0;
     with p do
          for i:=1 to m do
              for j:=1 to n do
                  for lat:=1 to min (m-i,n-j) do
                      for rot:=0 to lat-1 do
                          begin
                               mere:=calc;
                               if mere>meremax then
                                  begin
                                       meremax:=mere;
                                       pmax:=p
                                  end
                          end;
     with pmax do
          begin
               writeln('Pomii alesi au coordonatele:');
               writeln(i:4,j+rot:4,',',i+rot:4,j+lat:4,',',
               i+lat:4,j+lat-rot:4,',',i+lat-rot:4,j:4);
               scrie; {tiparirea livezii initiale}
               l[i,j+rot]:=-1;l[i+rot,j+lat]:=-1;
               l[i+lat,j+lat-rot]:=-1;l[i+lat-rot,j]:=-1;
               scrie {dupa culesul pomilor alesi (-1)}
          end
end.
```

Observaţie: Procedurile care seamănă cu altele folosite în probleme anterioare (citeste, scrie), au fost doar enunţate.

Scanned with OKEN Scanner

5.9 (V)

Se citesc elementele mai multor mulţimi formate din numere întregi. Să se afişeze reuniunea lor (sunt cel mult 20 de mulţimi, fiecare mulţime având cel mult 100 de elemente).

R 5.9

Rezolvarea va construi reuniunea pe măsură ce sunt citite elementele mulţimilor, astfel încât să nu se memoreze elementele tuturor mulţimilor.

```
program reun;
type vector =array[1..2000] of integer;
var a,b,c:vector;
    n,m,i,j,k,n_mult:integer;

function mai_este(x:integer;m:vector;i:integer):boolean;
var j:integer;
begin
     mai_este:=false;
     for j:=1 to i do
         if m[j]=x then mai_este:=true
end;
procedure citeste(var m:vector;var card:integer);
var i,j:byte;
begin
     write('Cate elemente are multimea? ');
     readln(card);
     for i:=1 to card do
         repeat
              write('Elementul ',i,':');
              readln(m[i]);
         until not mai_este(m[i],m,i-1);
end;
begin
     write('Numar multimi=');
     readln(n_mult);
     writeln('Prima multime');
     citeste(a,n);
     c:=a;k:=i;
     for j:=2 to n_mult do begin
         writeln('Multimea a ',j,'-a');citeste(b,m);
         for i:=1 to m do
             if not mai_este(b[i],c,k) then begin
                k:=k+1;
                c[k]:=b[i]
             end
         end;
     writeln('reuniunea este:');
     for i:=1 to k do write(c[i]:4);
     readln
end.
```

Observaţie: pentru scrierea în ordine crescătoare a elementelor reuniunii, fie se aplică un algoritm de ordonare a vectorului c înainte de a-l scrie, fie se inserează la locul lui fiecare element la care s-a ajuns (mutându-se elementele următoare câte o poziţie la dreapta în vector).

Scanned with OKEN Scanner

5.10 (IX – C)

Se citeşte un şir de caractere. Să se stabilească dacă şirul dat reprezintă o atribuire simplă de numere reale în limbajul PASCAL.

Observaţie: prin atribuire simplă am definit atribuirea în care expresia aritmetică din membrul drept nu conţine paranteze.

R 5.10

Se vor avea în vedere: sintaxa instrucţiunii de atribuire, corectitudinea de formă a expresiei aritmetice din membrul drept (operanzi şi operatori), fără a putea face vreo apreciere cu privire la compatibilitatea operanzilor sau cu privire la alegerea identificatorilor de variabile (de exemplu, să nu fie cuvinte rezervate):

```
program atribuire;
const op=['+','-','*','/'];
      cif=['0'..'9'];
      lit=['a'..'z','A'..'Z','_'];
var a:string;
    corect:boolean;
procedure spatii;          {Elimina spatiile de la inceput}
begin
     while a[1]=' ' do delete(a,1,1)
end;
function numar:boolean;{Verif.daca este o secventa numerica corecta}
var np:byte;               {np=numarul de puncte zecimale}
begin
     np:=0; spatii;
     numar:=false;
     repeat
           while a[1] in cif do begin
                 numar:=true; delete(a,1,1)
           end;
           if (a[1]='.')and(np=0) then begin
              np:=1; delete(a,1,1)
           end
     until not (a[1] in cif);
     spatii
end;
function variabila:boolean; {Verif. daca e id. de variabila corect}
begin
     variabila:=false;spatii;
     if not(a[1] in lit) then exit;
     variabila:=true;
     while a[1] in lit+cif do delete(a,1,1);
     spatii
end;
begin
    write('Introduceti sirul:');readln(a);
    a:=a+#13; corect:=true;
    if not variabila then corect:=false;
    spatii;
    if copy(a,1,2)<>':=' then corect:=false;
    delete(a,1,2);
    if a[1] in ['-','+'] then delete(a,1,1);
```


Scanned with OKEN Scanner

```
    while corect do begin
          if not(variabila or numar)then corect:=false;
          if (a[1] in ['*','/'])and not(a[2] in op)
             then delete(a,1,1)
    end;
    if a<>#13 then writeln('Aceast sir nu este o atribuire')
              else writeln('Aceast sir este o atribuire')
end.
```

5.11 (VII)

Se citeşte de la tastatură o expresie aritmetică fără paranteze, pe care o considerăm corectă. Operanzii sunt numere reale fără semn (cele cu semn vor fi scrise între paranteze), iar operatorii sunt +, -, *, / cu semnificaţiile cunoscute. Să se afişeze rezultatul expresiei.

R 5.11

Algoritmul funcţionează pe următorul principiu: dintre două operaţii consecutive, sigur una se efectuează: cea de prioritate maximă sau dintre două operaţii consecutive de aceeaşi prioritate se execută prima. Utilizăm procedurile str, val, delete şi insert pentru a efectua chiar pe expresia dată fiecare operaţie prioritară din secvenţa formată din primele două operaţii detectate în şir. Din cauza duplicităţii operatorului '-' (operator de semn – unar – şi operator de scădere – binar –) vom înlocui în expresie caracterul '-' cu alt caracter, de exemplu '_'.

```
program expresie_aritmetica;
const op=['+','_','/','*'];
var ex:string; op1,op2:char;
    i,poz1,poz2:byte;
procedure eroare;
begin
    writeln('Expresie eronata'); halt
end;
function prior(c:char):byte;
{stabileste prioritatea operatorului c}
begin
    if (c='+')or(c='_') then prior:=1
                        else prior:=2
end;
procedure spatii;
{elimina spatiile introduse de procedura str}
begin
    while (pos(' ',ex)<>0) do
          delete(ex,pos(' ',ex),1)
end;

procedure detect_op(s:string;var o:char;var p:byte);
var i:byte;   {detecteaza primul operator din s si pozitia sa}
begin
    p:=0;
    for i:=1 to length(s) do
        if s[i] in op then begin
           o:=s[i]; p:=i; exit
        end
end;
```


Scanned with OKEN Scanner

```
procedure calc(o:char;p:byte);
{substituie in expresie operatia cu rezultatul sau}
var t1,t2,rez:real;
    s:string;
    i,j:integer;
    cod:integer;
begin
     i:=p-1; j:=p+1;
     while (i>0)and not (ex[i] in op) do i:=i-1;
     val(copy(ex,i+1,p-i-1),t1,cod);
     if cod<>0 then eroare;
     while (j<=length(ex))and not (ex[j] in op) do j:=j+1;
     val(copy(ex,p+1,j-p-1),t2,cod);
     if cod<>0 then eroare;
     case ex[p] of
        '*':rez:=t1*t2;
        '/':rez:=t1/t2;
        '+':rez:=t1+t2;
        '_':rez:=t1-t2
     end;
     str(rez:6:2,s);
     delete(ex,i+1,j-i-1);
     insert(s,ex,i+1)
end;
begin
     write('Expresia aritmetica:');
     readln(ex);
     for i:=2 to length(ex) do
         if ex[i]='-'then ex[i]:='_';
     repeat
           spatii;
           detect_op(ex,op1,poz1);
           detect_op(copy(ex,poz1+1,length(ex)),op2,poz2);
           {detecteaza o eventuala a doua operatie}
           poz2:=poz2+poz1;
           if poz2<>poz1 then
              if prior(op1)>=prior(op2) then calc(op1,poz1)
                                        else calc(op2,poz2)
           else calc(op1,poz1)
     until poz2=poz1;
     writeln(ex)
end.
```

Recomandăm celor interesaţi să aprofundeze problema analizei expresiilor aritmetice algoritmul Rutihauser.

5.12 (X)

Să se realizeze tabelul de valori trigonometrice ale unui unghi dat (în grade). Programul trebuie să permită o execuţie repetată până ce se apasă tasta ESC.

R 5.12

Se utilizează o funcţie de conversie din grade în radiani şi câte o funcţie pentru fiecare funcţie trigonometrică, aceasta întorcând câte un string: valoarea formatată sau expresiile '+INFINIT','–INFINIT':

Scanned with OKEN Scanner

```
program val_trigonometrice;
uses crt;
const fmic=0.000001;
type tip_func=(si,co,tg,ct);
var i:tip_func;
    u:integer;
    c:char;
function rad(x:integer):real;
begin
     rad:=pi*x/180
end;
function trigo(x:real;t:tip_func):string;
var s:string;semn:char;
begin
     case t of
          si:str(sin(x):10:5,s);
          co:str(cos(x):10:5,s);
          tg:if abs(cos(x))<fmic then begin
                if sin(x)>0 then semn:='+'
                            else semn:='-';
                s:=' '+semn+'INFINIT'
             end
             else str(sin(x)/cos(x):10:5,s);
          ct:if abs(sin(x))<fmic then begin
                if cos(x)>0 then semn:='+'
                            else semn:='-';
                s:=' '+semn+'INFINIT'
             end
             else str(cos(x)/sin(x):10:5,s)
   end;
   trigo:=s
end;
begin
     clrscr;
     write(' UNGHIUL   NORMAL') {val.unghi si val.normalizata}
     writeln('     SINUS      COSINUS     TANGENTA   COTANGENTA');
     repeat
          Write('Unghiul dorit:');readln(u);
          gotoxy(1,wherey-1);delline;
          write(u:8,u mod 360:8);
          for i:=si to ct do write(trigo(rad(u),i):12);
          writeln;
          write('Apasati o tasta (ESC pentru sfarsit)'#13);
          c:=readkey;
          delline
     until c=#27
end.
```

5.13 (VIII)

Fiind dat un an (după Hristos), să se listeze calendarul acelui an.

R 5.13

Fiind dat anul, se determină ziua în care a căzut 1 ianuarie în acel an. S-a notat Duminica cu 0, Luni cu 1, etc. Formula de calcul a zilei de 1 ian. este dată în

Scanned with OKEN Scanner

„A Collection of Programming Problems and techniques", H.A.Maurer, M.R. Williams, Prentice Hall, 1972:

```
program calendar;
uses crt;
const
  IAN=1; FEB=2; MAR=3; APR=4;  MAI=5;  IUN=6;
  IUL=7; AUG=8; SEP=9; OCT=10; NOV=11; DEC=12;
  l:array[1..12]of string[10]=('Ianuarie','Februarie','Martie',
    'Aprilie','Mai','Iunie','Iulie','August','Septembrie',
    'Octombrie','Noiembrie','Decembrie')
  blank=' ';
var luna,an,zi:integer;
function legal(an:integer):boolean;
begin
   legal:=(an>0);
end;
procedure prima_zi (var an,zi:integer);
var y:integer;
begin
   repeat
     write('Anul pentru care doriti calendarul:  ');
     readln(an);
     writeln;
     if not legal(an) then
        writeln('Nu se poate face un calendar pentru acest an.');
   until legal(an);
   y:=an-1;
   zi:=(36+y+(y div 4)-(y div 100)+(y div 400)) mod 7;
end;
function bisect(an:integer):integer;
{se scoate 1 daca anul este bisect, 0 altfel}
begin
  if (an mod 4=0) and (an mod 100<>0) then bisect:=1
  else
      if an mod 400=0 then bisect:=1
      else bisect:=0;
end;
function nr_zile(luna,an:integer):integer;
{da numarul de zile din luna}
begin
  case luna of
    IAN,MAR,MAI,IUL,AUG,OCT,DEC: nr_zile:=31;
    FEB: nr_zile:=28+bisect(an);
    APR,IUN,SEP,NOV: nr_zile:=30;
  end;
end;
procedure antet_luna(luna,an:integer);
{tipareste antetul cu numele lunii,anul si saptamana}
begin
   write(l[luna],' ');
   writeln(an); writeln;
   writeln('  D  L  M  M  J  V  S');
end;
```


Scanned with OKEN Scanner

```
procedure scrie_datele(zi,lung_luna:integer);
{Se scriu zilele lunii in forma de calendar; se tiparesc intai
blancuri in prima saptamana, apoi zilele lunii, revenind la prima
pozitie la fiecare inceput de saptamana}
var
  date:integer;
begin
   if zi>0 then write(blank:zi*3);
   for date:=1 to lung_luna do begin
      write(date:3);
      if (date+zi) mod 7=0 then writeln;
   end;
   if (lung_luna+zi) mod 7<>0 then writeln; writeln
end;
procedure scrie_luna(luna,an,zi:integer); {scrie datele unei luni
      din an; zi este prima zi a lunii: Duminica-0, Luni-1, etc.}
begin
  antet_luna(luna,an);
  scrie_datele(zi,nr_zile(luna,an));
end;
begin {Programul principal}
  clrscr;
  prima_zi(an,zi);
  for luna:=IAN to DEC do begin
     scrie_luna(luna,an,zi);
     readkey; {luna ramane afisata pana se apasa o noua tasta}
     zi:=(zi+nr_zile(luna,an)) mod 7;
  end;
end.
```

5.14 (**VIII – C**, Concurs ACM, 1992; Finala Timiş, 1995)

Băcanul din oraşul dumneavoastră încearcă o nouă metodă de afişare a preţurilor produselor pe care le are în stoc. În loc de a marca pentru fiecare articol preţul (în lei), etichetele lui au preţuri comparative faţă de alte produse. De exemplu, untul poate fi etichetat *margarină + 100*, margarina poate fi etichetată *cafea + 111*; câteva articole sunt marcate chiar cu preţul respectiv.

Aţi aflat că marfa lui este foarte ieftină şi vreţi să faceţi o listă cu preţurile produselor, în maniera obişnuită (vezi ieşirea pentru exemplul de mai jos). Scrieţi un program care să realizeze acest lucru.

Intrare:

Fiecare set de date constă dintr-o succesiune de linii de forma:

```
lapte = zahar - 125
faina = 225
zahar = faina + 10
cafea = ceai
#
```

În legătură cu datele de intrare, se mai presupun următoarele:

– toate articolele au nume formate din maxim 10 litere mici;

– există cel mult 100 de articole;

– liniile de intrare dau fie direct preţul unui articol, fie îl consideră egal cu preţul altui produs eventual + sau – un număr întreg de lei;

Scanned with OKEN Scanner

– fiecare articol apare o singură dată în stânga semnului =;
– ultima linie conţine numai caracterul #.

Ieşire:

Pentru fiecare set de date trebuie să tipăriţi o listă ordonată alfabetic, a produselor menţionate, împreună cu preţurile lor. Dacă preţul unui produs nu poate fi dedus din fişierul de intrare, daţi-l ca fiind blank.

De exemplu, ieşirea pentru exemplul de mai sus este:

```
cafea = blank
ceai  = blank
faina = 225
lapte = 110
zahar = 235
```

R 5.14

Problema prezintă un grad sporit de dificultate, ea presupunând un algoritm de analiză sintactică şi semantică rudimentară a fiecărei linii prin mai multe parcurgeri ale listei de preţuri, până la obţinerea unei liste stabile (care nu se mai modifică de la o parcurgere la alta).

Analiza fiecărei linii presupune:

– separarea produsului de evaluat (membrul stâng al egalităţii);

– selectarea tipului evaluării în fiecare caz:

a) preţul direct al produsului (caz în care se înlocuieşte valoarea produsului respectiv într-o tabelă de preţuri);

b) produsul relativ la care se face evaluarea şi eventual valoarea de adăugat/scăzut (caz în care se caută în tabelă preţul produsului);

Algoritmul de principiu este următorul:

```
Citirea liniilor;
Evaluarea produselor_tip_a
Repeta
      Evaluarea produselor_tip_b_evaluabile
Pana cand nr_evaluari=0
Clasarea produselor neevaluate (pret-->blank)
Ordonarea perechilor <produs=pret>
Tiparirea perechilor <produs=pret>
```

S-a folosit artificiul marcării în lista de produse a produselor deja evaluate prin adăugarea unui caracter '=' la sfârşitul numelui produselor respective.

```
program bacan;
type sir=string[10];
     lista=array[1..100] of sir;
var prod,pret:lista;  {prod-lista de produse, pret-lista de preturi}
    la_fel:boolean;s:string;
    n,i,j:byte;valoare,cod:integer;
function posibil(var p:sir):boolean;       {daca expresia p poate fi
     evaluata in functie de preturile existente, ea va fi calculata}
var prod_relativ:sir;
    pret_relativ:integer;
    i,k:byte;dif:integer;
```


Scanned with OKEN Scanner

```
begin
     k:=pos('+',p)+pos('-',p);
     if k<>0 then prod_relativ:=copy(p,1,k-1)+'='
             else prod_relativ:=p+'=';
     posibil:=false; i:=1;
     while (i<=n)and(prod[i]<>prod_relativ) do i:=i+1;
     if i>n then exit;
     posibil:=true;
     if k<>0 then val(copy(p,k,length(p)),dif,cod)
             else dif:=0;
     val(pret[i],pret_relativ,cod);
     str(pret_relativ+dif,p)
end;
begin
     i:=1;       { citirea datelor cu evaluari imediate de preturi}
     write('Linia ',i:2,':');readln(s);
     while s<>'#' do begin
           j:=pos('=',s);
           prod[i]:=copy(s,1,j-1);
           pret[i]:=copy(s,j+1,length(s)-j);
           val(pret[i],valoare,cod);
           if cod=0 then prod[i]:=prod[i]+'=';       {o data evaluat,
                    produsul se identifica prin <nume> urmat de = }
                i:=i+1;
                write('Linia ',i:2,':');readln(s);
           end;
     n:=i-1;
{evaluari ulterioare repetate pana ce nu se mai poate face niciuna }
     repeat la_fel:=true;
           for i:=1 to n do
               if prod[i][length(prod[i])]<>'=' then
                  if posibil(pret[i]) then begin
                     prod[i]:=prod[i]+'='; la_fel:=false
                  end
     until la_fel;
     for i:=1 to n do        { clasarea produselor neevaluate }
         if prod[i][length(prod[i])]<>'=' then pret[i]:='=blank';
     for i:=1 to n-1 do      { ordonarea alfabetica a produselor }
         for j:=1 to n-i do
             if prod[j]>prod[j+1] then begin
                s:=prod[j];prod[j]:=prod[j+1];prod[j+1]:=s;
                s:=pret[j];pret[j]:=pret[j+1];pret[j+1]:=s
             end;
     for i:=1 to n do writeln(prod[i],pret[i]); {scriere rezultate}
end.
```

5.15 (VI – C, Olimpiada Naţională Arad, 1993; Concurs PACO, 1995)
Se dă un text de cel mult 255 de caractere (litere mici şi spaţii).

a) Găsiţi cea mai lungă secvenţă în care literele sunt ordonate alfabetic;

b) Determinaţi cea mai lungă secvenţă simetrică (care este aceeaşi citită de la stânga la dreapta şi invers);

c) Rescrieţi textul astfel încât cuvintele să fie ordonate alfabetic;

Scanned with OKEN Scanner

d) Construiţi o frază cu număr maxim de cuvinte (alese din textul dat) astfel încât litera finală a fiecărui cuvânt să fie aceeaşi cu prima literă a cuvântului următor şi cuvintele să nu se repete.

R 5.15

Propunem rezolvările pentru subpunctele (a),(b) şi (c), punctul (d) nefiind accesibil celor ce nu cunosc tehnica recursivităţii şi metoda **backtracking** (vezi Capitolul 8 unde acest punct se va relua în cadrul problemei 8.13).

Algoritmul de principiu pentru punctele (a) şi (b) este: se asociază fiecărui caracter din şir un număr care arată câte litere următoare sunt ordonate alfabetic (respectiv formează o secvenţă simetrică), determinându-se şirul de lungime maximă cu această proprietate. Algoritmul pentru punctul (c) este o ordonare de secvenţe, după ce fraza a fost descompusă în cuvinte.

Observaţie: În text pot exista şi spaţii, acestea fiind ignorate din punct de vedere alfabetic.

```
program text_;
const litere=['a'..'z']+[' '];
var t,aux:string; nr_cuv,i,j:byte;
    a:array[0..255]of byte;
    s:array[1..100]of string[100];
function poz_max:byte;
var j,max:byte;
begin
     max:=a[1]; poz_max:=1;
     for j:=2 to length(t)-1 do
         if a[j]>max then begin
            max:=a[j]; poz_max:=j
         end;
end;
function palindrom(ini,fin:byte):boolean;
var i:byte;
begin
     palindrom:=false;
     for i:=ini to fin do
        if t[i]<>t[fin-i+1] then exit;
     palindrom:=true;
end;
procedure punctul_a;
var gata:boolean;
   i,pas:byte;
begin
     write('Punctul a : ');
     for i:=1 to length(t)-1 do a[i]:=0;
     i:=1;
     repeat
          j:=i;gata:=false;
          repeat  {ignorarea spatiilor la criteriul alfabetic}
                pas:=1;
                while t[j+pas]=' ' do pas:=pas+1;
                if t[j]>t[j+pas] then gata:=true;
                a[i]:=a[i]+pas;j:=j+pas
          until gata;
          i:=i+a[i]
     until i>=length(t);
```


Scanned with OKEN Scanner

```
      i:=poz_max;
      writeln(copy(t,i,a[i]));
end;
procedure punctul_b;
begin
     write('Punctul b : ');a[0]:=1;
     for i:=1 to length(t)-1 do begin
         a[i]:=a[i-1];
         for j:=length(t)-2 downto i+a[i-1] do
             if palindrom(i,j) then begin
                a[i]:=j-i+1;
                j:=i+a[i-1]{break}
             end
     end;
     i:=poz_max;
     writeln(copy(t,i,a[i]));
end;
procedure punctul_c;
begin
     write('Punctul c : ');{sortarea prin inversiuni}
     for i:=1 to nr_cuv-1 do begin
         for j:=i+1 to nr_cuv do
             if s[i]>s[j] then begin
                aux:=s[i]; s[i]:=s[j]; s[j]:=aux
             end;
         write(s[i],' ')
     end;
     writeln(s[nr_cuv])
end;
begin
     write('Introduceti textul:');readln(t);
     for i:=1 to length(t) do
         if not(t[i] in litere) then begin
            writeln('Text incorect');halt
         end;
     t:=t+' '#13;
     punctul_a;
     punctul_b;
     nr_cuv:=0;
     while t<>#13 do begin          {fragmentarea textului in cuvinte}
           while t[1]=' ' do delete(t,1,1)
           nr_cuv:=nr_cuv+1;
           s[nr_cuv]:=copy(t,1,pos(' ',t)-1);
           delete(t,1,pos(' ',t));
           while t[1]=' ' do delete(t,1,1)
     end;
     punctul_c;
end.
```

5.16 (**IX**, Concurs ACM, 1992)

În vederea asigurării unei transmiteri cât mai exacte de informaţii de la un calculator la altul, transferul se efectuează pe grupe de câte 8 biţi (octeţi) astfel: ultimii 7 biţi reprezintă informaţia, iar primul bit se numeşte bit de paritate şi are valoarea 0 dacă suma celor 7 cifre binare de informaţie este un număr par sau 1 în

Scanned with OKEN Scanner

caz contrar. Dându-se un text ASCII format din caractere imprimabile cu coduri între 32 şi 127, să se scrie succesiunile de câte 8 biţi corespunzătoare caracterelor, fiecare succesiune de 8 biţi fiind urmată de un blanc.

Exemplu: pentru textul 'Hello! Bye.' se obţine secvenţa de octeţi:

01001000 01100101 01101100 01101100 01101111 00100001 10100000 01000010 11111001 01100101 00101110 corespunzând reprezentărilor binare ale valorilor 72, 101, 108, 108, 111, 33, 32, 66, 121, 101 şi 46.

R 5.16

Funcţia de conversie a unui caracter (de fapt a codului său ASCII) în secvenţa de 8 cifre binare alipeşte resturile împărţirilor succesive la 2 (conform algoritmului de conversie a numerelor din baza 10 în altă bază) la sfârşitul şirului *ch*, realizând astfel inversarea succesiunii de resturi. Tipul rezultatului funcţiei (string[8]) a fost „botezat" într-o declaraţie *type* deoarece nu se permit în antetul unui subprogram tipuri structurate anonime (afară de tipul string – cu care programul ar fi rulat la fel de corect).

```
program bit_de_paritate;
type octeti=string[8];
var linie:string;i:byte;
procedure err(a:string);                    {Tratarea erorilor}
begin
     writeln(a);readln;halt
end;
function octet(i:byte):octeti;       {Conversie zecimal-binara}
var j,k,n:byte;ch:octeti;
begin
     n:=ord(linie[i]);k:=0;ch:='';
     if(n>127)or(n<32)then err('Caracter nepermis');
     for j:=1 to 7 do begin
        {Transformarea cifrelor binare in caractere}
         ch:=chr(n mod 2+ord('0'))+ch;
         k:=k+n mod 2; n:=n div 2
     end;
     octet:=chr(k mod 2+ord('0'))+ch
end;
begin
     write('Textul;'); readln(linie);
     for i:=1 to length(linie) do write(octet(i),' ')
end.
```

5.17 (IX)

Să se scrie un program care emite câte o secvenţă melodică descrisă de o funcţie liniară, pătratică, respectiv iraţională, fiind date frecvenţele limită, pasul de incrementare a frecvenţei şi durata întregii secvenţe.

R 5.17

Cele trei funcţii se apelează ca parametru funcţional de către procedura *s_ func* care se ocupă de generarea secvenţei de sunete. Apelul procedurilor şi funcţiilor ca parametri de subprograme solicită folosirea directivei {$F+}.

Scanned with OKEN Scanner

```
program sunete;                          {$F+}
uses crt;
const n=3;a=0.01;b=3;c=100;
type functie=function(x:word):word;
var ini,fin,s:word;
    p:integer;ch:char;d:real;
function f1(x:word):word;
begin
     f1:=x+200
end;
function f2(x:word):word;
begin
     f2:=round(a*sqr(x)+b*x+c)
end;
function f3(x:word):word;
begin
     f3:=round(sqrt(x/a))+100
end;
procedure s_func(i,f,p:integer;d:real;func:functie);
begin
     if f<i then p:=-p;
     repeat
           sound(func(i)); delay(25);
           nosound; delay(25);
           i:=i+p
     until (f-i)*p/abs(p)<0;                {p/abs(p)=sgn(p)}
     delay(1000);
end;
begin
     repeat
           write('Frecventa initiala [0..100]:');readln(ini);
           write('Frecventa finala   [0..500]:');readln(fin);
           write('Pasul [1..4]:');readln(p);
           s_func(ini,fin,p,d,f1);
           s_func(ini,fin,p,d,f2);
           s_func(ini,fin,p,d,f3);
           writeln('Mai doriti o proba? (D/N) ');
           repeat
                 ch:=upcase(readkey)
           until ch in['D','N']
     until ch='N'
end.
```

5.18 (IX – C, Baraj IOI, Focşani, 1995)

Un călător are de parcurs o distanţă D, dată în Km. Cheltuielile datorate călătoriei sunt distribuite gradat în trepte astfel:

– pentru primii l_1 Km parcurşi în fiecare zi se percepe un cost c_1 per Km;

– pentru următorii l_2 Km parcurşi în aceeaşi zi se percepe un cost c_2 per Km, ş.a.m.d.

Fiecare zi de drum presupune un cost suplimentar egal cu un număr p dat.

Să se determine în câte zile trebuie efectuată călătoria şi câţi Km trebuie să fie parcurşi în fiecare zi, astfel încât costul total al călătoriei să fie minim. Se va furniza o soluţie şi costul total corespunzător.

Scanned with OKEN Scanner

Se dau la intrare:

– pe prima linie, distanţa D ($D < 10000$);

– pe a doua linie, numărul total K ($K < 100$) de trepte;

– pe următoarele K linii, câte o pereche de numere l_i, c_i (cu $c_i > c_{i-1}$, $2 \le i \le K$, $l_1 + l_2 + ... + l_k \le D$);

– pe ultima linie, costul suplimentar p. (A se vedea exemplul).

Se cer la ieşire pe ecran:

– numărul n de zile stabilit;

– distanţele parcurse în fiecare zi;

– costul total al călătoriei.

Exemplu:

Pentru datele de intrare:

```
26          (D)
4           (K)
5 10        (l1,c1)
4 15        (l2,c2)
10 21       (l3,c3)
10,30       (l4,c4)
40          (p)
```

se obţine o soluţie:

```
3                       (n)
9                       (prima zi)
9                       (a doua zi)
8                       (a treia zi)
435                     (cost total)
```

R 5.18

Numărul de Km corespunzători lungimii totale a drumului se pot distribui în mod egal în 1, 2, ... d zile, eventualii Km rămaşi nedistribuiţi aşezându-se fie într-o zi nouă, fie în oricare dintre zilele anterioare (după cum este mai convenabil din punctul de vedere al costului). Se reţine modul optim de distribuire (numărul de zile care minimizează costul) şi se tipăreşte soluţia corespunzătoare.

```
program cost_calatorie;
uses crt;
const dmax=10000; tmax=100;
type vector=array[0..dmax] of longint;
var a:vector; cost_min,cost:longint;
    l,c:array[1..tmax] of integer;
    i,j,t,k,p,d:integer;
procedure citeste;
var nume:string;
begin
     write('Distanta totala:');readln(d);
     write('Numar etape posibile:');readln(k);
     for i:=1 to k do begin
         writeln('Etapa ',i);
         write('Distanta si costul/km:'); readln(l[i],c[i])
     end;
     write('Cost suplimentar/zi:');readln(p);
end;
```


Scanned with OKEN Scanner

```
procedure costuri;
begin
     t:=0; a[0]:=0;
     for i:=1 to k do begin
         if t+1[i]>=d then 1[i]:=d-t;
         for j:=t+1 to t+1[i] do a[j]:=a[j-1]+c[i];
         t:=t+1[i];
     end;
end;
function min(a,b:longint):longint;
begin
     min:=(a+b-abs(a-b))div 2;
end;
procedure rezolva;
begin
     costuri;
     cost_min:=maxlongint;
     for t:=1 to d do begin       {distribuirea km uniform cate t/zi}
         cost:=d div t*(p+a[t]);
         if d mod t>0 then   {optimizarea asezarii km nedistribuiti}
            cost:=cost+min(p+a[d mod t],a[d mod t+t]-a[t]);
            if cost<cost_min then begin
               cost_min:=cost;
               k:=t;
            end;
     end;
end;
procedure scrie;
begin
  if a[k+d mod k]-a[k]<p+a[d mod k] then begin
     for i:=1 to d div k-1 do
         writeln('Ziua ',i,': dist. parcursa ',k);
     writeln('Ziua ',d div k,': distanta parcursa ',k+d mod k)
  end
  else begin
       for i:=1 to d div k do
           writeln('Ziua ',i,': distanta parcursa ',k);
       writeln('Ziua ',d div k+1,':distanta parcursa ',d mod k)
  end;
  writeln('Cost total:',cost_min)
end;
begin
     citeste;
     rezolva;
     scrie
end.
```

$$\min(a,b)=\frac{a+b-|a-b|}{2}$$

Observaţie: Ideea rezolvării a fost dată în concurs de Gheorghioiu Ovidiu din Alba-Iulia.

5.19 (**VII**, Baraj IOI, Bucureşti, 1995)

Se consideră mulţimea $S=\{a^2+b^2 \mid a,b\in\{0,1,\ldots,n\}\}$.

Se consideră şirurile formate din elemente ale mulţimii S, astfel încât diferenţa oricăror doi termeni consecutivi este constantă (notată cu r). Să se afişeze toate şirurile cu un număr maxim de termeni, precizând r şi primul termen.

Scanned with OKEN Scanner

Exemplu:

Pentru $n = 4$ avem $S = \{0, 1, 2, 4, 5, 8, 9, 10, 13, 16, 17, 18, 20, 25, 32\}$

```
r=1:        sirul  0 1 2
r=2:        sirul 16 18 20
r=3:        sirul 2 5 8
r=4:        sirul 0 4 8 si sirul 1 5 9 13 17
r=5         sirul 0 5 10
```

.....

Observaţie: n ($n \leq 10$) se citeşte de la tastatură şi se afişează raţia şi primul termen.

Pentru exemplul de mai sus se va afişa:

```
4 1
```

(prin care se identifică şirul 1 5 9 13 17)

R 5.19

Problema revine la generarea şirurilor formate din numere „bipătrate" aflate în progresie aritmetică şi apoi determinarea celor de lungime maximă. Pentru aceasta, fie se generează şirul tuturor numerelor bipătrate până la $n^2 + n^2 = 2n^2$ şi se caută subşirurile cu proprietatea din enunţ, fie se generează toate progresiile aritmetice cu raţii între 1 şi $d\max = 2*n^2$, termenul iniţial putând lua valori între 0 şi *raţia* – 1 (de la raţia în sus regăsindu-se termenii progresiilor deja generate) cât timp fiecare nou termen este un număr „bipătrat". În programul de faţă s-a preferat a doua metodă:

```
program bipatrate;
uses crt;
const max=10; dmax=2*max*max;
var bip:array[0..dmax] of boolean;
    n,ini,ratia,lmax,l:integer;
procedure generez;
var i,j:integer;
begin
   fillchar(bip,dmax,false); {initializarea vect. bip cu val false}
     for i:=0 to n do
         for j:=i to n do bip[i*i+j*j]:=true
end;
function lung_sir(i:integer):integer;
begin
     l:=0;
     while (i<=dmax) and bip[i] do begin inc(l); i:=i+ratia end;
     lung_sir:=l
end;
begin
     write('n=');readln(n);
     lmax:=0;
     generez;
     for ratia:=1 to n*n*2 do
         for ini:=0 to ratia-1 do
             if lung_sir(ini)>lmax then lmax:=l;
     for ratia:=1 to n*n*2 do
         for ini:=0 to ratia-1 do
             if lung_sir(ini)=lmax then
                writeln(ratia,' ',ini);
end.
```


Scanned with OKEN Scanner

Probleme propuse

5.20 (V)

Să se scrie o funcţie de calcul a celui mai mare divizor comun a două numere întregi pozitive. Funcţia nu va folosi decât operaţii de scădere.

5.21 (X)

Să se construiască o funcţie *comb(n, k)* care determină numărul combinărilor de n obiecte luate câte k ($0 \le k \le n \le 100$).

5.22 (V)

Să se scrie o procedură *putere(k, tab)* care pentru un k ($30 \le k \le 50$) întreg, calculează în tabloul tab cifrele componente ale numărului 2^k.

5.23 (V)

Se citesc numărătorii şi numitorii a n fracţii de numere întregi; să se afişeze fracţia de numere întregi obţinută prin înmulţirea fracţiilor date.

Indicaţie: a se urmări rezolvarea problemei 5.2.

5.24 (VI)

Se citeşte un şir de n ($n \le 10000$) numere reale. Fiind date diverse valori lui k, să se afişeze suma primilor k termeni ai şirului. Execuţia programului se încheie când se furnizează de la tastatură $k = 0$.

Indicaţie: Se poate lucra cu o procedură *suma* ce calculează suma primilor k termeni deoarece ideea de a calcula sumele parţiale de la 1, 2, ..., n într-un alt vector cu n componente eşuează din cauza lipsei spaţiului de memorie necesar alocării celui de-al doilea vector.

5.25 (VIII)

Să se afişeze tabelul de semne pentru o funcţie de forma $(ax + b) * (cx + d)$ cu a, b, c, d reale date.

5.26 (IX)

Să se rezolve în mulţimea numerelor reale ecuaţia $a * x^4 + b * x^2 + c = 0$ cu a, b, c reale date, folosind un subprogram de rezolvare a ecuaţiei de gradul 2.

5.27 (IX)

Se citeşte o expresie logică fără paranteze, formată din operanzi notaţi cu câte o literă şi operatori logici (codificaţi & – *şi*, | – *sau*, ! – *not*). Se dau apoi valorile operanzilor şi se cere să se afişeze valoarea logică a expresiei. Pentru aceeaşi expresie, programul trebuie să permită evaluări repetate.

Exemplu: pentru expresia $! a \mid b \,\&\, c$ se dau

$a = 1$ (sau true), $b = 0$ (sau false), $c = 1$ şi se afişează $! a \mid b \,\&\, c = 0$

$a = 0, b = 1, c = 0$ şi se afişează $! a \mid b \,\&\, c = 1$.

Observaţie: Operatorii logici au priorităţi diferite: prior(!)>prior(&)>prior(|)

Scanned with OKEN Scanner

5.28 (XI)

Să se determine punctele de extrem şi natura acestora (minim sau maxim) pentru o funcţia $\cos(x) - 2/3 * x^3$ pe un interval dat.

Indicaţie: Funcţia şi derivata sa se definesc ca funcţii Pascal, se caută pe subintervale rădăcinile derivatei şi – dacă există – se evaluează funcţia în punctele respective şi în capetele intervalului. Pentru detalii, a se vedea [8].

5.29 (V – C, Concursul de informatică, faza finală, Timiş, 1995)

Doi prieteni stabilesc un cod de transmitere a mesajelor lor astfel: în fiecare cuvânt, o consoană urmată de o vocală se va înlocui cu litera *S*, iar după o vocală care are înaintea sa în textul original o consoană, se adaugă consoana precedentă şi litera *O*.

a) Se consideră un text dat de la tastatură, care conţine cuvinte date numai cu litere mari şi separate prin spaţii; se cere să se tipărească textul codificat;

b) Se dă un text şi se cere textul original, a cărui codificare o reprezintă textul dat.

Exemplu:

a) MASCA ---> SAMOSSACO

b) SAMOSSACO ---> MASCA

5.30 (IX – C, Cluj, Etapa judeţeană, 1995)

Fie $n \leq 3000$ un număr natural. Se cere să se determine un multiplu al său în a cărui scriere în baza 10 să intervină numai cifrele 0 şi 1.

5.31 (IX – C, Constanţa, Etapa judeţeană, 1995)

Să se scrie un program care, pentru m, n ($m < n$) numere naturale nenule date, găseşte şi afişează numerele naturale $q_1, q_2, \ldots, q_r$ astfel încât:

i) $q_{k-1} \mid q_k$, pentru orice k ($2 \leq k \leq r$);

ii) $\frac{m}{n} = \frac{1}{q_1} + \frac{1}{q_2} + \ldots + \frac{1}{q_r}$.

Scanned with OKEN Scanner

VI. UNITĂŢI DE PROGRAM

Probleme rezolvate

6.1 (V)

Să se scrie pe ecran, pentru orice tastă apăsată, codul sau codurile ASCII ale caracterului asociat tastei respective.

R 6.1

Utilizăm funcţia *readkey* din unitul CRT pentru a „culege" caracterele transmise la apăsarea unei taste. Curent, apăsarea unei taste transmite un caracter (tipăribil –> 'A' sau netipăribil –> CR); sunt însă şi taste care transmit două caractere succesive, dintre care primul are obligatoriu codúl 0 (de exemplu tasta HOME transmite caracterele #0 şi #75). Codul ASCII asociat caracterului transmis la apăsarea tastei ESC este 27. În condiţii de sintaxă extinsă {$X+} orice funcţie poate fi folosită şi ca procedură care execută corpul de instrucţiuni scris la definirea funcţiei, fără a întoarce un rezultat. De exemplu funcţia *readkey*, apelată ca procedură (instrucţiune de sine stătătoare), are ca efect aşteptarea unei taste, deci este echivalentă cu instrucţiunea compusă *repeat {instrucţiune vidă} until keypressed*.

```
{$X+}
program taste;
uses crt;
var c:char;
begin
     writeln('Apasati o tasta (ESC pentru sfarsit):');
     repeat
          c:=readkey; write('#',ord(c));
          if c=#0 then
             begin
                  c:=readkey; writeln(' #',ord(c))
             end
          else writeln
     until c=#27;
     writeln('Gata');
     readkey
end.
```

6.2 (VIII)

Se dă un caracter de la tastatură. Să se scrie un program care permite mişcarea caracterului pe ecran prin acţionarea tastelor săgeţi în orice direcţie, fără a permite ieşirea din ecran. De asemenea se cere posibilitatea de comutare între modul mişcare *cu urmă* şi *fără urmă* prin apăsarea tastei ENTER, comutarea fiind anunţată şi printr-un sunet scurt.

R 6.2

Se precizează caracterul care va fi mişcat, se şterge ecranul, se desenează caracterul într-o poziţie oarecare (ale cărei coordonate se generează cu funcţia

Scanned with OKEN Scanner

random) şi apoi se efectuează succesiv citiri de la tastatură (cu funcţia *tasta*) şi scrieri pe ecran (cu procedura *mişcă*), până la apăsarea tastei ESC. Procedura *mişca* desenează caracterul spaţiu pe vechea poziţie (pentru comutatorul *urma =false*) şi caracterul dat pe poziţia curentă. Menţionăm că apăsarea tastelor săgeţi generează o secvenţă de două caractere (testaţi cu programul asociat rezolvării problemei precedente), primul fiind caracterul cu codul #0, iar al doilea caracterul corespunzător unuia din codurile (71–Home, 72–↑, 73–Page Up, 75–←, 77–→, 79–End, 80–↓, 81–Page Down). Apăsarea tastei ENTER generează un caracter cu codul 13, iar ESC dă caracterul cu codul 27. S-au considerat accesibile coloanele ecranului între 1 şi 80 şi liniile între 1 şi 24, linia 25 fiind rezervată indicaţiilor de acţionare a tastelor.

```
program miscare;
uses crt;
type directii=(e,ne, n,nv, v,sv,s,se,vid);
const
   dx:array[directii]of shortint=(1, 1, 0,-1,-1,-1,0,1,0);
   dy:array[directii]of shortint=(0,-1,-1,-1, 0, 1,1,1,0);
   dir:array[#71..#81]of directii=(nv,n,ne,vid,v,vid,e,vid,sv,s,se);
{ataseaza tastelor #0#71..#0#81 directiile geograf. corespunzatoare}
var c,p:char;x,y:byte;
    a:directii;
    urma:boolean;
procedure nici_o_tasta; {anuleaza orice tasta apasata anterior}
begin
     while keypressed do c:=readkey
end;
procedure misca(d:directii);     {deplaseaza caracterul pe ecran}
begin
     if not urma then begin
        gotoxy(x,y); write(' ')
     end;                    {cu conditia sa ramana in ecran}
     if ((dx[d]>=0)and(x<79)or(dx[d]<=0)and(x>1))
        and((dy[d]>=0)and(y<24)or(dy[d]<=0)and(y>1)) then begin
        x:=x+dx[d]; y:=y+dy[d]
     end;
     gotoxy(x,y);write(p)
end;
function tasta:boolean;    {citeste si interpreteaza o tasta apasata}
begin                      {intoarce false numai daca s-a apasat ESC}
     tasta:=false;
     nici_o_tasta;
     repeat c:=readkey until c in [#0,#13,#27];
     if c=#27 then exit;
     tasta:=true;
     case c of
          #0: begin c:=readkey; misca(dir[c]) end;
          #13:begin
                   sound(200);delay(100);nosound;
                   urma:=not urma              {comutarea switch-ului}
              end
     end
end;
```


Scanned with OKEN Scanner

```
begin
     write('Caracterul plimbator:');
     readln(p)
     clrscr;
     gotoxy(1,25);
     write('<- -> ^ v miscare     ENTER urme   ESC iesire');
     x:=1+random(79);y:=1+random(24);
     misca(vid);
     repeat until not tasta
end.
```

6.3 (V)

Să se scrie pe ecran mai multe texte în ferestre diferite ale căror colţuri (stânga-sus şi dreapta-jos) se dau de la tastatură. Un text se introduce până nu mai încape în fereastra definită sau până se apasă ESC, moment în care se permite definirea altei ferestre. Ultima linie din ecran se va utiliza ca fereastră de dialog între utilizator şi calculator.

R 6.3

Amintim că procedura *window* activează o zonă de ecran la care se vor referi toate celelalte proceduri şi funcţii de lucru cu ecranul din unitul CRT (în afară de o altă procedură *window*). Ultimul caracter din fereastră nu se poate scrie cu procedura write fără a împiedica efectul de defilare în sus a textului (doar atribuind caracterul de scris unei variabile cu adresă absolută în memoria video); de aceea se va permite umplerea ferestrei până la poziţia relativă (*maxy*, *maxx*) :

```
program ferestre;
uses crt;
type fereastra=record
                    ps,pd:record
                                x,y:byte
                          end;
                    maxx,maxy:integer
               end;
var f:fereastra;
    c:char;
procedure win_dialog;
begin
     textcolor(white);
     textbackground(black);
     window(1,25,80,25);
     clrscr
end;
procedure win_text(a:fereastra);
begin
     textcolor(white);
     textbackground(black);
     window(a.ps.x,a.ps.y,a.pd.x,a.pd.y);
     clrscr
end;
begin
     textbackground(white);
     clrscr;
```


Scanned with OKEN Scanner

```
repeat
      with f do begin
           win_dialog;
           repeat
                write('Coltul stanga-sus (linie coloana):');
                readln(ps.y,ps.x)
           until (ps.x in [1..80])and(ps.y in [1..24]);
           repeat
                write('Coltul dreapta-jos (linie coloana):');
                readln(pd.y,pd.x)
           until (pd.x in [1..80])and(pd.y in [1..24]);
           maxx:=pd.x-ps.x+1;maxy:=pd.y-ps.y+1;
           if (maxx<0) or (maxy<0) then write('Coord. eronate')
           else begin
                win_text(f);
                repeat
                     c:=readkey;
                     if c=#13 then write(#13#10)
                     else if c>=#32 then write(c)
                          else write(#7)  {sunet}
                until (wherex=maxx)and(wherey=maxy)or(c=#27);
                write(#7); win_dialog
           end;
           write('Apasati o tasta (ESC pentru iesire):');
           c:=readkey
      end
until c=#27
end.
```

Observaţii: Programului i se pot aduce o serie de îmbunătăţiri: verificări astfel încât să nu se permită suprapuneri de ferestre (trebuie memorate toate ferestrele în *f:array[1..max]of ferestre* şi comparate apoi coordonatele noii ferestre cu coordonatele fiecărei ferestre anterioare), setarea culorilor de fond şi cerneală (*textcolor* şi *textbackground*) după dorinţă, etc. Dacă procedurile de lucru cu ferestre se aduc într-o formă utilă şi altor programe, nu mai este necesar ca ele să fie copiate de fiecare dată în acele programe; ele se pot „împacheta" separat într-un unit propriu (numit ferestre, de exemplu) care se va folosi în alte programe declarându-l obişnuit (*uses crt, ferestre,...;*). Procedurile şi funcţiile din unitul propriu pot fi apelate ca şi cum ar fi definite chiar în program. Atenţie: numele fişierului în care se salvează unitatea ferestre trebuie să se numească tot ferestre (în urma compilării pe disc se crează, alături de ferestre.pas, şi ferestre.tpu care trebuie să fie accesibil tuturor programelor ce folosesc unitatea ferestre). A se vedea şi [2].

6.4 (IX)

Să se scrie un unit util programelor cu adunări şi scăderi de numere pozitive foarte mari (cu cel mult 255 de cifre), numerele fiind scrise într-o aceeaşi bază *b* dată. Redactaţi un program care calculează suma şi diferenţa a două numere scrise în baze date (cel mult 200).

Observaţie: pentru bazele mai mari decât 10 se vor folosi caracterele 'A', 'B', etc.

R 6.4

Se implementează funcţii de conversie din caracter_cifra în valoarea numerică a cifrei şi invers (*cif* şi *car*), funcţia de validare a scrierii unui număr în

Scanned with OKEN Scanner

baza *b* (*verif*), funcţia de complementare a unui număr în baza *b* – complementul faţă de *b* – utilă operaţiei de scădere (*neg*), funcţii sumă şi diferenţă (*adun* şi *scad*). De observat că nu se precizează în interfaţă decât funcţiile şi procedurile care vor fi utile altor programe, nu şi cele de „bucătărie" ale unitului.

```
unit oper_str;          {se va salva sub numele oper_str.pas}

interface
function verif(x:string;b:byte):boolean;
function adun(x,y:string;b:byte):string;
function scad(x,y:string;b:byte):string;

implementation
function cif(c:char):integer;
begin
     cif:=ord(c)-48-7*byte(c>'9');
end;
function car(n:byte):char;
begin
     car:=chr(n+48+7*byte(n>9))
end;
function verif;
var i:byte;
begin
     verif:=false;
     for i:=1 to length(x) do
         if (cif(x[i])<0)or(cif(x[i])>=b) then exit;
     verif:=true
end;
function adun;
var rez:string;
    s,t:integer;
    i:byte;
begin
     for i:=length(x)+1 to length(y) do x:='0'+x;
     for i:=length(y)+1 to length(x) do y:='0'+y;
     t:=0;rez:='';
     for i:=length(x) downto 1 do begin
         s:=cif(x[i])+cif(y[i])+t;
         t:=byte(s>=b);
         s:=s-b*t;
         rez:=car(s)+rez
     end;
     if t=1 then rez:='1'+rez;
     adun:=rez
end;
function neg(x:string;b:byte):string;
var i:byte;
begin
     for i:=1 to length(x) do
         x[i]:=car(b-1-cif(x[i]));
     neg:=adun(x,'1',b)
end;
```


Scanned with OKEN Scanner

```
function scad;
var rez:string;
    i:byte;
begin
     for i:=length(x)+1 to length(y) do x:='0'+x;
     for i:=length(y)+1 to length(x) do y:='0'+y;
     rez:=adun(x,neg(y,b),b);delete(rez,1,1);
     while(rez<>'')and(rez[1]='0')do delete(rez,1,1);
     scad:=rez
end;
end.
```

Unitul se va folosi în orice program util, în cazul nostru în programul:

```
program operatii;
uses oper_str;
var a,b:string;ba,bb:byte;
     o:char;
procedure lit(var a:string);
var i:byte;
begin
     for i:=1 to length(a) do a[i]:=upcase(a[i])
end;
begin
     repeat
          write('a='); readln(a); lit(a);
          write('in baza '); readln(ba)
     until verif(a,ba);
     {... analog pentru b in baza bb}
     if bb<>ba then halt;
     write('Operatia (+ - ) :'); readln(o);
     write(a,'(',ba,')',o,b,'(',bb,')=');
     case o of
          '+': write(adun(a,b,bb));
          '-': if a>=b then write(scad(a,b,bb))
                       else write('nr.negativ');
        else write('Operatie eronata ')
     end;
     writeln('(',bb,')')
end.
```

6.5 (IX)

Să se organizeze un vector de numere reale împreună cu dimensiunea sa (ca date) construind procedurile de citire şi scriere a elementelor sale, precum şi o funcţie care verifică dacă şirul este ordonat descrescător şi-l ordonează în caz contrar (ca metode) într-un obiect. Să se ordoneze descrescător un şir de numere întregi date folosind obiectul definit.

R 6.5

Prezentăm pe un tip simplu de problemă modul de lucru obiectual, folosind un obiect definit chiar în program (de obicei acestea se definesc în unităţi separate, în program definindu-se eventual descendenţi ai acestora) şi cuprinzând numai metode statice. Prezentarea amănunţită a lucrului cu obiecte, noţiuni de descendenţă şi polimorfism nu se încadrează în obiectivele acestei lucrări (vezi [3], [2]):

Scanned with OKEN Scanner

```
program obiect;
type sir=
        object
              x:array[1..100]of real;
              n:byte;
              procedufe citeste;
              procedure scrie;
              function ordo:boolean;
        end;
procedure sir.citeste;
var i:byte;
begin
     write('Cate elemente are sirul? ');
     readln(n);
     for i:=1 to n do begin
         write('Elementul nr.',i,' al sirului:');
         repeat
              {$I-}readln(x[i]){$I+};
         until ioresult=0
     end
end;
procedure sir.scrie;
var i:byte;
begin
     writeln;
     for i:=1 to n do write(x[i]:10:3);
     writeln
end;
function sir.ordo;
var i,j:byte; aux:real;
begin
     ordo:=true;
     for i:=1 to n-1 do
         for j:=1 to n-i do
             if x[j]<x[j+1] then begin
                     aux:=x[j];x[j]:=x[j+1];
                     x[j+1]:=aux;ordo:=false
             end
end;
var s:sir;
begin
     s.citeste;
     if not s.ordo then s.scrie
     else writeln('Sirul este deja ordonat')
end.
```

6.6 (VIII)

Să se scrie un program care citeşte de la tastatură un număr reprezentând frecvenţa iniţială a unei game, altul ce reprezintă durata gamei în întregime şi permite alegerea dintr-un meniu a uneia dintre secvenţele muzicale clasice (gamă sau arpegiu, majoră sau minoră, armonică sau melodică). Pentru fiecare set de date se generează secvenţa muzicală corespunzătoare, atât ascendentă cât şi descendentă. Programul trebuie să poată fi rulat în mod repetat.

Scanned with OKEN Scanner

R 6.6

Folosirea la fiecare apel a procedurilor de generare sunet (*sound*), de pauză (*delay*) şi de întrerupere sunet (*nosound*) pentru fiecare sunet, ar încărca inutil programul. De aceea redactăm procedura sunet cu doi parametri (frecvenţă şi durată) şi o folosim pentru fiecare tip de secvenţă muzicală. Teoria muzicală prezintă o gamă ca un interval împărţit în 12 subintervale de frecvenţe între care există o proporţie ($\sqrt[12]{2}$), intervale numite semitonuri. Notele gamei împart intervalul în subintervale neegale, unele dintre ele de câte un ton (*t* = două semitonuri), altele de câte un semiton (st = un semiton), astfel:

– gama majoră are treptele 1- *t* - 2 - *t* - 3 - *st* - 4 - *t* - 5 - *t* - 6 - *t* - 7 - *st* - 8,
– gama minoră are treptele 1- *t* - 2 - *st* - 3 - *t* - 4 - *t* - 5 - *st* - 6 - *t* - 7 - *t* - 8,
– gama armonică are treptele 1- *t* - 2 - *st* - 3 - *t* - 4 - *t* - 5 - *st* - 6 - *t*+*st* - 7 - *st* - 8 ,
– gama melodică are treptele 1- *t* - 2 - *st* - 3 - *t* - 4 - *t* - 5 - *t* - 6 - *t* - 7 - *st* - 8.

S-a folosit un vector *gama* care memorează reperele notelor muzicale în cele 12 noduri ale intervalului; de exemplu pentru gama majoră prima notă (tonica) = 0, a doua = 2 (distanţa de 2 semitonuri faţă de tonică), a treia = 4, a patra = 5, etc. Tabloul *g* memorează cele 8 tipuri de game şi arpegii prezentate în meniul din programul principal.

Atragem atenţia asupra folosirii funcţiilor şi procedurilor imbricate (definite în alte proceduri) prin care se poate accentua faptul că aceste subprograme sunt utile procedurii – *sunete* – din care fac parte. Ele puteau fi însă definite şi în afara procedurii, cu condiţia ca definirea să se facă înaintea procedurii *sunete*. Dacă, de exemplu, se doreşte emiterea unui sunet în programul principal, actuala structură a programului nu permite aceasta. Variabilele globale *f*, *d*, *o* sunt general valabile, mai puţin *f* şi *d* care nu pot fi active în procedura *sunete* unde au fost redefinite ca parametri de intrare. Apelul s-a făcut prima dată chiar pentru valorile *f* şi *d* din programul principal, apoi apelurile s-au făcut pentru diverse valori ale lui *f*.

```
program game;
uses crt;
type gama=array[1..7] of real;
const g:array[1..8]of gama=((2,4,5,7,9,11,12),(4,7,12,0,0,0,0),
        (2,3,5,7,8,10,12),(3,7,12,0,0,0,0),(2,3,5,7,8,11,12),
        (3,7,12,0,0,0,0),(2,3,5,7,9,11,12),(3,7,12,0,0,0,0));
var f,d,o:integer;
procedure sunete(i:byte);
var n_note,j:byte;
function putere(e,b:real):real;   {b=baza,e=exponentul}
begin
      putere:=exp(e*ln(b))        {formula de calcul pentru b^e}
end;
procedure sunet(f,d:integer);
begin
     sound(f);
     delay(d);
     nosound;
     delay(d)
end;
```


Scanned with OKEN Scanner

```
begin
     if i mod 2=1 then n_note:=7
                  else n_note:=3;
     sunet(f,d);
     for j:=1 to n_note do
         sunet(round(f*putere(1/12,putere(g[i,j],2))),d);
     for j:=n_note downto 1 do
         sunet(round(f*putere(1/12,putere(g[i,j],2))),d);
     sunet(f,d);
end;
begin
     clrscr;
     write('Precizati frecventa initiala(20..2000):');
     readln(f);
     write('Precizati durata unei game in secunde:');
     readln(d);
     d:=d*1000 div 16;
     writeln('1) Gama majora    2) Arpegiu major ');
     writeln('3) Gama minora    4) Arpegiu minor');
     writeln('5) Gama armonica  6) Arpegiu armonic');
     writeln('7) Gama melodica  8) Arpegiu melodic');
     writeln('9) Gata');
     repeat
          repeat
               gotoxy(5,20);delline;
               write('Optiunea:');readln(o)
          until o in [1..9];
          if o<>9 then sunete(o)
     until o=9
end.
```

6.7 (V)

Să se genereze aleator un labirint format din elemente 0 şi 1 (1 = culoar, 0 = zid). Să se simuleze căutarea unui drum care străbate labirintul de la prima la ultima linie printr-un program interactiv. Utilizatorul va căuta un punct de intrare în linia 1 a labirintului şi va încerca apoi să înainteze prin labirint, la fiecare încercare calculatorul semnalându-i dacă poate înainta sau nu în direcţia respectivă.

R 6.7

Se folosesc procedurile de lucru cu ferestre-ecran (*window*) din unitul CRT. Rolul fiecărei proceduri este explicat în program sub formă de comentarii.

```
program labiri;
 uses crt;
 const y:array[1..8]of shortint=(0,-1,-1,-1, 0, 1,1,1);
       x:array[1..8]of shortint=(1, 1, 0,-1,-1,-1,0,1);
       tmax=100000;
 type tablou=array[0..25,0..25] of byte;
 const n=20;
 var a:tablou;
     l,c,i,j:byte;
     timp:integer;
     g:byte;
```


Scanned with OKEN Scanner

```
procedure generare; {genereaza aleator elemente 0 si 1 in tabloul a}
var i,j:byte;
begin                              {tabloul ramane bordat cu elemente 2}
     randomize;window(1,1,50,25);
     for i:=0 to n+1 do
         for j:=0 to n+1 do a[i,j]:=2;
     for i:=1 to n do
         for j:=1 to n do begin
             a[i,j]:=random(2);
             gotoxy(2*j,i);write('. ')
         end;
     a[1,1+random(n)]:=1
end;
procedure asteapta(var r:char);       {asteapta apasarea unei taste}
const m=['1'..'4','6'..'9','Q','q'];
begin
     window(60,7,80,25);gotoxy(5,6);
     repeat
           repeat
                 delay(100);
                 timp:=timp+1
           until keypressed;
           r:=readkey
     until r in m
end;
procedure zid(l,col:byte);              {deseneaza un element de zid}
begin
     g:=g+1;
     window (1,1,50,25);
     gotoxy(2*col,l);Write('#');
     sound(100);delay(100);nosound;
     window (51,24,80,25);
     writeln(' Aveti ',g,' greseli !!')
end;
procedure caut_intrare(var col:byte);  {gasire intrare in linia 1}
var m:char;
begin
     window(51,24,80,25);
     repeat
           write('Coloana intrarii '); readln(col);
           if a[1,col]=0  then zid(1,col)
     until a[1,col]=1
end;
procedure desen(l,c:byte;car:char); {deseneaza un element de culoar}
begin                    {pozitie precedenta-*  pozitia curenta-@}
  window(1,1,50,25);
  gotoxy(2*c,l); write(car);
  if (l=n)and(car='@') then begin
      window(51,24,80,25);
      writeln('  Ai reusit !!!!!');
      sound(500);delay(1000);nosound;
      readln;halt
  end
end;
```


Scanned with OKEN Scanner

```
procedure avansare(l,c:byte);     {incercari de miscare in labirint}
const key:array[1..8]of char=('6','9','8','7','4','1','2','3');
var r:char;i:byte;
begin
     asteapta(r);
     while (upcase(r)<>'Q')and (timp<tmax) do begin
           i:=1;
           while key[i]<>r do i:=i+1;
           if a[l+y[i],c+x[i]]=2 then begin
              sound(100);delay(100);nosound
           end
           else
               if a[l+y[i],c+x[i]]=0 then zid(l+y[i],c+x[i])
               else begin
                    desen(l,c,'*');
                    l:=l+y[i];c:=c+x[i];
                    desen (l,c,'@');
               end;
           asteapta(r)
     end
end;
begin
     clrscr;
     generare;
     timp:=0; g:=0; window (60,7,80,25);
     writeln('@=pozitia curenta'#10#13'#=perete *=culoar'#10#13);
     writeln ('7   8   9');writeln(' NV N NE');
     writeln ('4 v   e 6');writeln (' SV S SE');
     writeln ('1   2   3');
     writeln; writeln('q   QUIT');
     caut_intrare(c); desen(1,c,'@'); avansare(1,c);
     sound(120);delay(1000); nosound
end.
```

6.8 (VII)

Să se deseneze în centrul ecranului un pătrat de latură dată şi cercul circumscris acestuia.

R 6.8

Procedura *init* iniţializează placa grafică încărcând driverul grafic (programul de interfaţă între comenzile de acces la memoria grafică şi particularităţile hard ale plăcii grafice detectate) şi abordând unul dintre modurile grafice de care este capabilă placa. Programele-driver se găsesc de obicei într-un director special BGI (*Borland Graphics Interface*) care trebuie specificat împreună cu calea sa (de obicei C:\BP\BGI în afara situaţiei când fişierele **.bgi* sunt chiar în directorul curent, caz în care calea poate fi şirul vid). Funcţia *ajust*, ajustează dimensiunile verticale care altfel vor apărea diferite (deşi numeric egale) de dimensiunile orizontale. Raportul aspect-imagine *n*1/*n*2 şi dimensiunile maxime ale ecranului sunt specifice fiecărui mod şi fiecărei plăci grafice. Deci nu desenăm nimic în valori absolute, ci totul va fi calculat în funcţie de *getmaxx*, *getmaxy* (funcţii) şi de *n*1, *n*2 furnizate de procedura *getaspectratio*. Schimbarea raportului astfel încât să

Scanned with OKEN Scanner

fie 1/1 (cu *setaspectratio*) este în general periculoasă conducând la deformarea altor elemente grafice (cercuri, poligoane,etc.). Comutarea între modul grafic şi cel text se face cu procedurile *restorecrtmode* şi *setgraphmode*, informaţia video nefiind conservată între comutări.

```
program patrat;
uses graph;
var 1,n1,n2:word;
    dg,mg:integer;
    xc,yc:integer;
procedure init;
var cale:string;
begin
 write('Calea driverului grafic (pentru C:\BP\BGI apasati ENTER):');
  readln(cale);
  if cale='' then cale:='C:\BP\BGI';
  detectgraph(dg,mg); initgraph(dg,mg,cale);
  if graphresult<>grok then begin
         writeln('Cale incorecta'); halt
  end;
end;
function ajust(v:integer):integer;
begin
     ajust:=trunc(n1/n2*v)
end;
begin
     init;
     restorecrtmode;
     write('Latura patratului [10..',trunc(getmaxy/sqrt(2)),']:');
     readln(1);
     setgraphmode(mg); getaspectratio(n1,n2);
     xc:=getmaxx div 2; yc:=getmaxy div 2;
     moveto(xc-1 div 2,yc-ajust(1) div 2);
     linerel(1,0);linerel(0,ajust(1));
     linerel(-1,0);linerel(0,-ajust(1));
     circle(xc,yc,trunc(1/sqrt(2)));
     settextstyle(sansseriffont,horizdir,1 div 100);
     settextjustify(righttext,bottomtext);
     outtextxy(xc-1 div 2,yc-ajust(1) div 2,'A');
     settextjustify(lefttext,bottomtext);
     outtextxy(xc+1 div 2,yc-ajust(1) div 2,'B');
     settextjustify(lefttext,toptext);
     outtextxy(xc+1 div 2,yc+ajust(1) div 2,'C');
     settextjustify(righttext,toptext);
     outtextxy(xc-1 div 2,yc+ajust(1) div 2,'D');
     readln; closegraph
end.
```

6.9 (VII)

Să se deseneze pe ecran *n* cercuri echidistante de rază dată, cu centrele situate pe o linie orizontală imaginară dusă prin mijlocul ecranului, scriind în centrul fiecăruia un caracter dat. Cum se modifică programul dacă se cere ca cercurile să fie tangente?

Scanned with OKEN Scanner

R 6.9

Se calculează distanţa dintre centre astfel încât acestea să fie la aceeaşi distanţă atât între ele cât şi de marginile ecranului. Dacă se doreşte prevenirea ieşirii din ecran, se va pune condiţia suplimentară $r < d$ (deşi elementele de desen din afara ecranului nu conduc la întrerupere cu eroare, ci pur şi simplu nu se văd). Mărimea caracterelor se poate determina proporţional cu raza cercului, folosind de exemplu procedura *settextstyle (Defaultfont, Horizdir, r div 20).*

```
program cercuri;
uses graph;
var cale:string; gd,gm:integer;
    c:char;
    n,i,r,d,mx,my:word;
procedur init ;{vezi programul anterior}
.....
begin
     init;
     mx:=getmaxx;
     my:=getmaxy;
     restorecrtmode;
     write('Cate cercuri doriti sa desenati? ');
     readln(n);
     write('Raza fiecaruia exprimata in pixeli (20,',my div 2,'):');
     readln(r);
     write('Carcterul central:');
     readln(c);
     setgraphmode(gm);
     d:=mx div (n+1);              {distanta dintre centre}
     for i:=1 to n do circle(i*d,my div 2,r);
     settextjustify(Centertext,Centertext);
     for i:=1 to n do outtextxy(i*d,my div 2,c);
     readln;
     closegraph
end.
```

Pentru cerinţa ca cercurile să fie tangente, nu se mai cere raza ci se calculează distanţa în funcţie de dimensiunile ecranului.

```
     repeat
          write('Cate cercuri doriti sa desenati? ');
          readln(n)
     until n*8<mx;          { diametrul minim=8, pentru vizibilitate}
     r:=mx div n div 2;
     write('Carcterul central:');
     readln(c);
     setgraphmode(gm);
     for i:=1 to n do circle((2*i-1)*r,my div 2,r);
```

6.10 (IX – C)

Se cunosc coordonatele vârfurilor unui triunghi în sistemul de axe de coordonate carteziene (numere întregi). Să se realizeze reprezentarea triunghiului la dimensiunile întregului ecran, astfel încât să încapă în întregime şi să nu apară deformat.

Scanned with OKEN Scanner

R 6.10

Se pune problema determinării dreptunghiului în care se înscrie triunghiul, translatarea figurii în origine şi reprezentarea ei proporţională. Variabilele *minx*, *miny*, *maxx*, *maxy* calculează coordonatele reale ale colţurilor acestui dreptunghi. Factorii de proporţionalitate ai dreptunghiului real, faţă de dreptunghiul-ecran sunt *getmaxx*/lăţimea_reală şi *getmaxy*/înălţimea_reală, factorul de redimensionare a dreptunghiului fiind cel mai mic dintre cei doi factori. Deplasarea figurii în colţul din stânga jos al ecranului se face translatând toate punctele cu (*minx*, *miny*) în sensul negativ al axelor. În procedura *drawpoly* s-a transmis repetat primul punct (p[4]:=p[1]) pentru a se închide poligonul determinat de cele 3 puncte:

```
program triunghi;
uses graph;
type punct=record
                  x,y:integer;
          end;
var dg,mg:integer; cale:string;
    maxx,maxy,minx,miny,k:real;
    p:array[1..4]of punct;
    i:byte;
function max(a,b:real):real;
begin
     if a>b then max:=a
            else max:=b
end;
function min(a,b:real):real;
. . . . .
procedure init;
. . . . .
begin
     for i:=1 to 3 do begin
         writeln('Dati coordonatele punctului ',i);
         write('Abscisa:');readln(p[i].x);
         write('Ordonata:');readln(p[i].y)
     end;
     minx:=min(p[1].x,min(p[2].x,p[3].x));
     miny:=min(p[1].y,min(p[2].y,p[3].y));
     maxx:=max(p[1].x,max(p[2].x,p[3].x));
     maxy:=max(p[1].y,max(p[2].y,p[3].y));
     if (maxx<>minx)and(maxy<>miny) then begin
        k:=min(getmaxx/(maxx-minx),getmaxy/(maxy-miny));
        if (p[1].x=0)and(p[2].x=0)and(p[3].x=0)then write(#7);
        for i:=1 to 3 do
            with p[i] do begin
                 x:=trunc(k*(x-minx));
                 y:=trunc(getmaxy-k*(y-miny));
            end;
        init;cleardevice;
        p[4]:=p[1];
        drawpoly(4,p); readln
     end;
     closegraph
end.
```

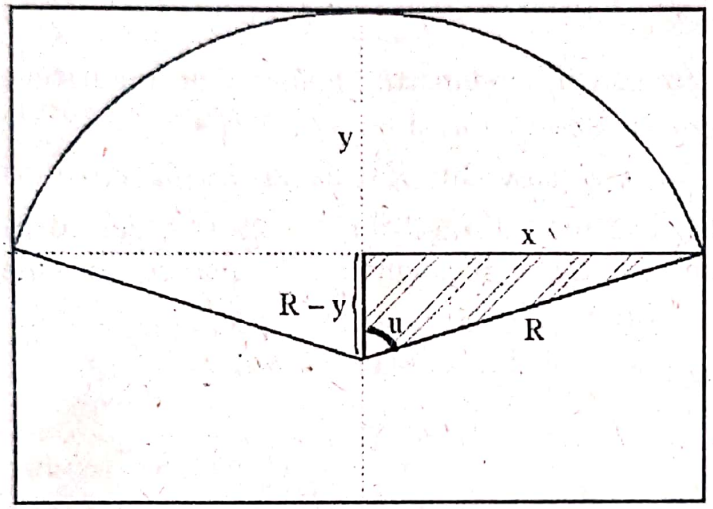

Scanned with OKEN Scanner

6.11 (IX)

Să se deseneze, folosind procedurile *arc*, *circle*, *pieslice* şi *floodfill*, un ochi căprui cu iris negru pe un fond de ecran negru, desenul fiind dimensionat la întregul ecran.

R 6.11

Calculul coordonatelor centrului, razei şi unghiurilor trigonometrice corespunzătoare arcelor ce formează conturul ochiului se deduc din următoarea figură în care $x = getmaxx$ div 2 şi $y = getmaxy$ div 2:



$$R^2 = x^2 + (R-y)^2$$

$$R = \frac{x^2 + y^2}{2y}$$

$$u = arctg\left(\frac{x}{R-y}\right)$$

```
program ochi;
uses graph;
var dg,mg:integer;
    x,y,r,u,yo:longint;
procedure init;
var cale:string;
begin
     write('Calea driverului grafic(pentru C:\BP\BGI apasati CR):');
     readln(cale);
     if cale='' then cale:='C:\BP\BGI';
     detectgraph(dg,mg); initgraph(dg,mg,cale);
     if graphresult<>grok then
     begin writeln('Cale incorecta'); halt end
end;
begin
     init;
     setbkcolor(black);
     cleardevice;
     x:=getmaxx div 2;y:=getmaxy div 2;
     r:=((sqr(x)+sqr(y))div(2*y));
     u:=round(arctan(x/(r-y))*180/pi);
     setcolor(white);
     arc(x,r,90-u,90+u,r);
     arc(x,2*y-r,270-u,270+u,r);
     setfillstyle(solidfill,white);
     floodfill(10,y,white);
     setcolor(brown);
     setfillstyle(solidfill,brown);
```


Scanned with OKEN Scanner

```
      pieslice(x,y,0,360,y div 3 * 2);
      setcolor(black);
      setfillstyle(solidfill,black);
      pieslice(x,y,0,360,y div 6);
      readln;
      closegraph
end.
```

6.12 (X)

Să se deseneze un poligon regulat, ştiind numărul laturilor şi raza cercului circumscris (exprimată în pixeli).

R 6.12

Se calculează punctele de pe cercul circumscris poligonului regulat cu formulele trigonometrice $x = x\_centru + r * \cos(i * u)$ şi $y = y\_centru + r * \sin(i * u)$ cu $i \in \{0,...,n-1\}$, unghiul u fiind a n-a parte din 360 de grade; de remarcat că ordonata unui punct se calculează relativ la colţul din stânga sus, deci dacă y este coordonata grafică, de fapt ea se va recalcula totdeauna cu formula $y = getmaxy - y$.

```
program poligon;
uses graph;
type punct=record
                  x,y:integer;
          end;
var r,dg,mg:integer;
    p:array[0..100]of punct;
    i,n:byte;
procedure init;
.............
begin
     write('Numarul de laturi:');readln(n);
     write('Raza cercului:');readln(r);
     for i:=0 to n-1 do
         with p[i] do begin
              x:=getmaxx div 2+round(r*cos(i*2*pi/n));
              y:=getmaxy div 2-round(r*sin(i*2*pi/n))
         end;
     p[n]:=p[0];
     init;
     setcolor(white); setbkcolor(black);
     cleardevice;
     drawpoly(n+1,p);
     readln;
     closegraph
end.
```

6.13 (X)

Să se reprezinte – folosind funcţii şi proceduri grafice – un cadran de ceas analogic care indică ora şi minutul, funcţionând cu mişcarea de 1 minut la o secundă reală. El va indica o oră dată (oră minut) şi va continua să indice ora actualizată la fiecare secundă, ca şi cum ar fi trecut un minut.

Scanned with OKEN Scanner

R 6.13

```
program ceasuri;
uses graph,crt;
var  dg,mg,m,lm,lo,i:integer;
     s:string[2];
     n1,n2,mx,my:word;
     o,um,uo:real;
     {um, uo->unghiuri trig. corespunzatoare minutului, orei}
procedure init;
........
function ajust(y:real):integer;
begin
     ajust:=trunc(n1/n2*y)
end;
procedure cadran;
begin
     settextstyle(defaultfont,horizdir,3);
     settextjustify(centertext,centertext);
     circle(mx,my,my);
     for i:=1 to 12 do begin
         str(i,s);uo:=pi/6*(3-i);
         outtextxy(mx+trunc(lm*cos(uo)),my-ajust(lm*sin(uo)),s)
     end;
end;
procedure limbi;
begin
     setlinestyle(solidln,0,thickwidth);
     setwritemode(xorput);  {efectul de transparenta}
     repeat
          uo:=pi/6*(3-o);um:=pi/30*(15-m);
          {punere}
          line(mx,my,mx+trunc(lm*cos(um)),my-ajust(lm*sin(um)));
          line(mx,my,mx+trunc(lo*cos(uo)),my-ajust(lo*sin(uo)));
          delay(1000);
          {stergere}
          line(mx,my,mx+trunc(lm*cos(um)),my-ajust(lm*sin(um)));
          line(mx,my,mx+trunc(lo*cos(uo)),my-ajust(lo*sin(uo)));
          m:=(m+1)mod 60;o:=(o+1/60);if o>=12 then o:=o-12
     until keypressed;
end;
begin
     write('Introduceti ora '); readln(o);
     write('Introduceti minutul '); readln(m);
     init;getaspectratio(n1,n2);
     o:=o+m/60; m:=m mod 60;                    {ora si minutul}
     mx:=getmaxx div 2; my:=getmaxy div 2;      {centrul ecranului}
     lm:=my div 4*3; lo:=my div 3;              {lungimile limbilor}
     cadran;limbi;
     closegraph
end.
```

Exerciţiu: În loc să se citească ora de la tastatură încercaţi să o obţineţi folosind procedura *gettime* din unitul DOS.

Scanned with OKEN Scanner

6.14 (IX – C, Concurs ACM, 1994 – variantă)

Să se simuleze grafic mişcarea unei bile într-o cutie de dimensiuni date, cunoscându-se poziţia ei iniţială relativ la colţul din stânga jos al cutiei, direcţia iniţială de mişcare precizată prin valoarea cosinusului unghiului trigonometric al direcţiei în care pleacă bila şi semnul sinusului aceleiaşi direcţii, precum şi coordonatele şi raza unei găuri practicate pe suprafaţa cutiei. Bila se consideră punctiformă, iar mişcarea ei se simulează până la intrarea în gaură sau până la apăsarea unei taste.

R 6.14

Se vor economisi multe calcule aferente rezolvării matematice a problemei (cum ar fi detectarea gropii) diferenţiind prin culoare punctele găurii şi apelând funcţia *getpixel*. De aceea linia ce reprezintă poziţia bilei la fiecare moment se desenează punct cu punct (*putpixel*) şi nu cu ajutorul procedurilor de desenat linii (*line*, *lineto*, *linerel*). Se lucrează în coordonate reale şi se utilizează numai pentru desen nişte funcţii de conversie la coordonate grafice (funcţii care sunt specifice oricăror probleme de reprezentare grafică). Valoarea raportului de reprezentare (raport) se calculează astfel încât cel puţin una dintre dimensiuni să se reprezinte la dimensiunea ecranului (rezervând 1/10 din lăţimea ecranului pentru mesaje).

```
program bila;
uses crt,graph;
var xmax,ymax,r,x,y,c,s,xg,yg,raport:real;        {Datele problemei}
    nume_gr:string;dg,mg:integer;                 {Parametri grafica}
    asp1,asp2:word; ch:char;
function min(a,b:real):real;.....
procedure init;.....
function grx(a:real):integer; {Functii de conversie coord. grafice}
begin
     grx:=trunc(a*raport)
end;
function gry(a:real):integer;
begin
     gry:=getmaxy-trunc(a*raport/asp2*asp1)
end;
procedure impacte;
begin {Deseneaza drumul bilei controland impactul cu marginea}
     repeat
          while (x>=0)and(x<=xmax)and(y>=0)and(y<=ymax) do begin
                if getpixel(grx(x),gry(y))=red then exit;
                putpixel(grx(x),gry(y),white);
                x:=x+c/raport;y:=y+s/raport
          end;
          sound(500);delay(20);nosound;
          ch:=readkey;
          if ch=#27 then exit;
          if (x<=0)or(x>=xmax) then c:=-c;
          if (y<=0)or(y>=ymax) then s:=-s;
          x:=x+c/raport;y:=y+s/raport;
     until false
end;
```


Scanned with OKEN Scanner

```
begin
      write('Dimensiuni cutie:');readln(xmax,ymax);
      write('Coord.initiale:');readln(x,y);
      repeat
            write('Directia (cos):');readln(c)
      until abs(c)<=1;
      s:=sqrt(1-sqr(c));
      write('sin=',s:6:4,' cu + sau - ?');readln(ch);
      if ch='-' then s:=-s;
      write('Coord. groapa (x y raza):');readln(xg,yg,r);
      init;
      getaspectratio(asp1,asp2);
      raport:=min(getmaxx/10*9/xmax,getmaxy/ymax);
      settextjustify(centertext,centertext);
      outtextxy(grx(x),gry(y),'o');
      settextstyle(sansseriffont,vertdir,2);
      outtextxy(getmaxx div 20*19,getmaxy div 2,'Apasati CR sau ESC');
      setfillstyle(solidfill,red);
      fillellipse(grx(xg),gry(yg),grx(r),grx(r));
      rectangle(0,getmaxy,grx(xmax),gry(ymax));
      sound(500);delay(20);nosound;readln;
      impacte;readkey;
      closegraph
end.
```

6.15 (VII)

Se citeşte un text care trebuie scris orizontal pe întreaga lăţime a ecranului, o dată în jumătatea superioară şi încă o dată în cea inferioară. Să se reprezinte o rotiţă ce avansează parcurgând întreaga suprafaţă a ecranului, fără a rescrie conţinutul acestuia la fiecare nouă poziţie a roţii, înaintarea făcându-se cu ştergerea textului în prima jumătate a ecranului şi transparent (peste text) în cealaltă jumătate.

R 6.15

Se utilizează setul vectorial de caractere (vezi [3]) TriplexFont care poate fi redimensionat cu ajutorul procedurii *UserCharSize*. Deoarece imaginea punctuală a roţii poate fi memorată în cei 64K alocaţi variabilelor statice, vom utiliza procedurile *putimage* şi *getimage* folosind ca depozitar al imaginii tabloul memo.

```
program animatie;
uses graph,crt;
var i,j,x,dg,mg:integer;          {driver_grafic, mod_grafic}
    s:string[30];
    r:real;
    memo:array[1..1600]of byte;
procedure init;.......
procedure scrie;          {deseneaza textul la marimea ecranului}
var mx,dx,my,dy:word;
begin
     settextstyle(triplexfont,horizdir,0);
     my:=trunc(getmaxy/textheight(s)*4);
     setusercharsize(10,10,my,10);
```


Scanned with OKEN Scanner

```
     mx:=trunc(getmaxx/textwidth(s)*10);
     setusercharsize(mx,10,my,10);
     settextjustify(centertext,toptext);
     outtextxy(getmaxx div 2,0,s);
     outtextxy(getmaxx div 2,getmaxy div 2,s);
end;
procedure anim;
var pus:word;
begin
     j:=0;i:=-1;x:=0;
     pus:=normalput;
     while keypressed do readkey;
     repeat
          i:=i*(-1);x:=x+3*i;              {i=1 dus;i=-1 intors}
          while (x>=0)and(x<=getmaxx-21) do begin
               putimage(x,j,memo,pus);{desenare normala}
               sound(j+x+20);
               putimage(x,j,memo,xorput);{stergere}
               nosound;
               if keypressed then exit;
               x:=x+3*i   {avans orizontal câte 3 pixeli}
          end;
          j:=j+40;                  {avans vertical un rand}
          if j>getmaxy div 2 then pus:=xorput
     until j>getmaxy-20
end;
begin
     write('Textul:');readln(s);
     init;cleardevice;
     for i:=1 to 4 do circle (20,20,5*i-1);
     {40*40=1600 dimensiunea maxima a tabloului memo}
     getimage(0,0,40,40,memo);                {memorare desen}
     cleardevice;
     if s<>'' then scrie;
     anim;
     closegraph
end.
```

Probleme propuse

6.16 (V)

Să se definească o fereastră albă de dimensiuni date în colţul de sus al ecranului negru, care să poată fi mişcată în spaţiul disponibil, până la o poziţie considerată definitivă (la apăsarea tastei ENTER).

Indicaţie: Deplasarea ferestrei se face ştergând fereastra activă după o setare a culorii hârtiei în negru şi activând o nouă fereastră ce va fi ştearsă cu fondul setat în alb.

6.17 (VII)

Să se asocieze fiecărui caracter de pe primul rând al tastaturii (12 din cele 15 taste incluzând tastele cifre) un sunet al gamei şi să se folosească pe post de pian până la apăsarea tastei ESC.

Scanned with OKEN Scanner

6.18 (VI)

Să se scrie interactiv un text dat într-o fereastră definită de utilizator astfel încât acesta să poată acţiona cu efectul obişnuit tastele săgeţi, home, end, backspace şi delete.

Indicaţie: Codurile tastelor specificate sunt menţionate în rezolvarea problemei 6.2, iar codul tastei backspace este 8.

6.19 (VII)

Să se reprezinte pe ecran un caracter oarecare într-o poziţie arbitrară, caracterul începând să se deplaseze orizontal într-o mişcare de „du-te vino" între marginile ecranului. La apăsarea oricărei taste, caracterul îşi schimbă direcţia de mişcare (din orizontală în verticală şi invers). Execuţia se încheie când se apasă tasta ESC.

6.20 (VIII)

Să se reprezinte în centrul ecranului, în poziţie orizontală, un şarpe format din 9 caractere 'o' şi un caracter '@' (capul). Şarpele se mişcă orizontal. La apăsarea tastelor săgeţi, se poate modifica direcţia de mişcare şi sensul pentru „capul" şarpelui, inelele urmând traiectoria imprimată de cap. Execuţia se încheie când „capul" ajunge afară din ecran.

Indicaţie: Se vor folosi doi vectori (X şi Y, de exemplu) pentru memorarea poziţiei fiecărui segment din corpul şarpelui (X_i = coloana, Y_i = linia), o înaintare presupunând doar modificarea poziţiei capului şi preluarea fiecărui segment a poziţiei segmentului precedent:

```
for i:=10 downto 2 do
    begin
        x[i]:=x[i-1];y[i]:=y[i-1]
    end
```

6.21 (IX)

Se citesc coordonatele vârfurilor unui triunghi. Să se reprezinte pe ecran astfel încât originea axelor de coordonate să se afle în centrul ecranului şi axele să fie paralele cu marginile ecranului.

Indicaţie: Se vor adapta funcţiile de conversie la coordonate grafice (vezi problema 6.14).

6.22 (IX)

Să se reprezinte un cerc trigonometric realizând umplerea sectorului de cerc corespunzător fiecărui unghi trigonometric dat de la tastatură (în grade).

Programul trebuie să permită execuţii repetate, după dorinţa utilizatorului.

6.23 (VII)

Să se deseneze o scară cu n ($n > 0$) trepte identice, prima treaptă pornind din colţul stânga-jos al ecranului, iar ultima ajungând în colţul din dreapta sus.

Scanned with OKEN Scanner

6.24 (VIII-C)

Să se acopere suprafaţa grafică a ecranului cu n ($n > 0$) dreptunghiuri neegale (cel puţin una dintre dimensiuni să difere). Dar dacă prin dreptunghiuri neegale se înţelege ca ambele dimensiuni să difere?

6.25 (IX)

Folosind n puncte echidistante de pe cercul tangent la trei dintre marginile ecranului, să se deseneze pe rând toate poligoanele cu k laturi (n, k date, $n \leq k$) având vârfurile printre cele n puncte.

6.26 (IX)

Să se deseneze o dreaptă (având date coordonatele capetelor) permiţând rotirea ei trigonometrică în jurul unuia dintre capete la apăsarea unei taste.

6.27 (VI)

Să se acopere suprafaţa ecranului cu dreptunghiuri albe şi negre (ca o tablă de şah), numărul de dreptunghiuri pe verticală şi numărul de dreptunghiuri pe orizontală fiind date de la tastatură.

6.28 (VI)

Să se deseneze n cercuri concentrice echidistante, cercul exterior fiind tangent la marginile de sus şi de jos ale ecranului, iar cel interior având raza dată.

6.29 (VII)

Să se deseneze un număr nelimitat de cercuri concentrice, cercul exterior fiind tangent la marginile ecranului, astfel încât distanţele dintre cercuri să se micşoreze spre centru, tinzând la 0.

6.30 (VIII)

Să se reprezinte în fiecare colţ al ecranului câte un caracter @ înscris într-un cerc tangent la două dintre marginile ecranului grafic. Manevraţi dimensiunile caracterului şi cercului pentru obţinerea unui efect estetic.

6.31 (VIII)

Să se reprezinte un sistem de axe de coordonate cu originea în centrul ecranului şi să se deseneze puncte ale căror coordonate reale sunt citite de la tastatură, până la introducerea perechii de coordonate (0,0).

6.32 (IX)

Se citesc coordonatele a n puncte dintre care cel puţin trei sunt necoliniare. Să se scaleze (aleagând o proporţie între unitatea reală şi numărul de puncte grafice corespunzătoare) şi să se amplaseze convenabil axele de coordonate, astfel încât toate cele n puncte să fie reprezentate vizibil în ecran şi să nu existe „benzi" nefolosite (fără reprezentări de puncte) pe margini.

Scanned with OKEN Scanner

6.33 (VII)

Se generează aleator poziţiile a n ($n > 0$) buburuze pe ecran şi se controlează de la tastatură poziţia unui puişor care trebuie să mănânce toate buburuzele într-un timp dat (proporţional cu numărul de buburuze). Poziţia iniţială a puişorului este în colţul stânga-sus al ecranului.

Aceeaşi cerinţă pentru o râmă care trebuie să mănânce buburuzele.

Indicaţie: Buburuzele şi puişorul se reprezintă folosind caractere (eventual cu coduri ASCII mai mari decât 127), iar pentru mişcare se vor revedea problemele 6.2, 6.7 şi 6.20.

6.34 (VIII)

Să se scrie o unitate care să conţină proceduri de ordonare şi funcţii care verifică dacă un şir de n numere reale este ordonat (crescător respectiv descrescător), şi funcţii care furnizează maximul respectiv minimul dintr-un şir de n numere reale date.

a) Utilizaţi unitatea într-un program care citeşte un şir de numere întregi şi afişează prima secvenţă crescătoare şi ultima secvenţă descrescătoare de lungime maximă din şir.

Exemplu: pentru şirul: 12 3 –6 9 11 20 5 0.6 11 15 22 –1
se afişează –6 9 11 20 şi 20 5 0.6

b) Folosind metode statice, încercaţi împachetarea şi utilizarea acestora ca un singur obiect.

6.35 (IX)

Să se scrie un unit util programelor cu operaţii pe numere pozitive foarte mari (cu cel mult 255 de cifre), numerele fiind scrise în orice bază. Redactaţi un program care calculează suma, produsul sau diferenţa a două numere scrise în baze date (cel mult 200).

Indicaţie: În completarea procedurilor şi funcţiilor prezentate în rezolvarea problemei 6.4, se va realiza procedura de înmulţire cu o cifră care va fi folosită de către procedura de înmulţire cu un număr format din mai multe cifre, rezultatul înmulţirii cu o cifră fiind apoi adunat la rezultatul anterior deplasat cu o poziţie spre stânga (shift). Procedurile de conversie din baza 10 în altă bază şi din altă bază în baza 10 vor putea fi utilizate de o procedură de conversie dintr-o bază oarecare în alta. Conversia în baza 10 poate folosi formula:

$$\overline{x = x_1 x_2 \ldots x_n} = x_n + b\,(x_{n-1} + b\,(\ldots(x_2 + bx_1)\ldots)),$$

unde b este baza în care este scris numărul x. Conversia din baza 10 în altă bază este cea mai laborioasă dintre proceduri, realizarea împărţirilor succesive putând fi redusă la scăderi repetate.

Scanned with OKEN Scanner

VII. FIŞIERE

Probleme rezolvate

7.1 (IX)

Programele Pascal sunt de fapt texte salvate pe disc sub forma unor fişiere secvenţiale (text). Pentru un scurt program existent pe disc, sunteţi solicitaţi să-l afişaţi pe ecran caracter cu caracter, făcând o pauză de 0.5 secunde după fiecare caracter. Redactaţi programul, salvaţi-l sub numele *fişier.pas* şi apoi executaţi-l.

R 7.1

Vom citi câte un caracter din fişier cu procedura *read* şi-l vom afişa pe ecran. Pauza efectuată după fiecare scriere dă posibilitatea să urmărim efectul celor două caractere #13 şi #10 (EOLN) la apariţia lor pe ecran. Eroarea care poate să apară în mod normal la deschiderea fişierului, este datorată faptului că nu a fost găsit fişierul fizic *fişier.pas* în directorul curent.

```
program fisier;
uses crt;
var f:text;
    c:char;
begin
     assign(f,'fisier.pas');
     {$I-} reset(f) {$I+};
     if ioresult<>0 then writeln('Fisierul nu exista')
     else
         while not (eof(f) or keypressed) do
               begin
                    read(f,c);
                    delay(500);
                    write(c)
               end
     writeln
end.
```

7.2 (IX)

Se consideră următorul program:

```
program test;
var f:text;n:real;
begin
     assign(f,'fis.txt');
     reset(f);
     while not eof(f) do
           begin
                read(f,n);
                write(n:6:2)
           end;
     writeln;
     close(f)
end.
```


Scanned with OKEN Scanner

Să se anticipeze execuţia sa pentru următoarele forme de conţinut ale fişierului 'fis.txt':

```
linia 1: 12.6 79 -5
linia 2: 8
linia 3: 9 99
```

```
linia 1: 12.6      79 -5    (spatiere)
linia 2: 8
linia 3:                    (linie alba)
linia 4: 9 99
```

```
linia 1: 12.6  79  -5
linia 2: 8
linia 3: 9 99....(spatii)
```

```
linia 1: 12.6  79  -5
linia 2: 8 9 99
linia 3:
```

Găsiţi o justificare fiecărui caz în parte.

R 7.2

În primul caz este creat un fişier clasic (corect spaţiat, fără linii albe) şi rezultatele coincid cu conţinutul (diferă doar formatul de scriere al numerelor). În cazul următor (din dreapta), deşi apar spaţii şi linii intermediare albe, rezultatele execuţiei sunt aceleaşi, deoarece citirea numerelor din fişierele text se face prin interpretarea primei secvenţe găsite după separatori (' '– cu codul 32, TAB cu codul 9, EOLN – cu codurile 13 şi 10, BACKSPACE – cu codul 8 şi BELL – cu codul 7), secvenţa interpretată ţinând până la următorul separator. Deci interpretarea numărului 12.6 se opreşte exact pe spaţiul de după acest număr, următoarea citire ignorând toţi separatorii până la 79 şi interpretându-l pe acesta până la spaţiul următor. Aceeaşi explicaţie se poate da pentru linia albă, interpretarea lui 8 făcându-se până la separatorul EOLN (#13+#10), următoarea citire ignorând toţi separatorii (acest EOLN şi următorul). Exemplele trei şi patru generează însă o situaţie problemă prin maniera de lucru cu caracterul EOF (caracterul cu codul 26, botezat uneori ^Z deoarece se obţine de la tastatură apăsând simultan tasta CTRL cu Z – vezi problema 6.1) existent ca marcaj al sfârşitului de fişier. A apărut un zero din cauză că după citirea numărului 99 „nu se vede" sfârşitul de fişier, deci se încearcă interpretarea unui nou număr *n* care însă nu mai există şi încercarea eşuează întorcând în *n* valoarea 0. Recomandăm în asemenea situaţii testarea valorii funcţiei *seekeof* în loc de *eof*.

Observaţie: Ca să creaţi corect spaţiile de la sfârşitul liniilor, folosiţi editorul Norton în locul editorului Pascal.

7.3 (IX)

Se consideră un fişier text care conţine numere pe mai multe rânduri, numerele de pe acelaşi rând fiind despărţite între ele prin spaţii. Să se determine câte numere sunt pe fiecare linie.

Dar dacă numerele sunt despărţite între ele prin virgule?

R 7.3

Testarea funcţiei EOLN pentru fişierul dat ne oferă posibilitatea să sesizăm trecerea pe rând nou, moment în care se afişează numărul liniei curente şi valoarea contorului *n* de numere, după care se resetează valoarea acestui contor şi „se trece

Scanned with OKEN Scanner

efectiv pe rând nou" în fişier (de fapt se înaintează liniar în fişier până după caracterele #13#10) cu ajutorul procedurii *readln*. Pentru clarificare, creaţi fişierul fis.txt cu editorul mediului Turbo Pascal şi testaţi programul următor:

```
program nr_pe_linii;
var f:text; r:real;
    i,k,n:integer;
begin
     assign(f,'fis.txt');
     reset(f);
     n:=0;
     while not eof(f) do
           begin
                readln(f);
                n:=n+1;
           end;
     writeln('Fisierul are ',n,' linii.');
     close(f); reset(f);
     for k:=1 to n do
         begin
{*}            i:=0;
{*}            while not eoln(f) do
{*}                  begin
{*}                       i:=i+1;
{*}                       read(f,r);
{*}                  end;
{*}            readln(f);
{*}            writeln('Nr. de numere aflate pe linia ',k,': ',i);
         end;
     close(f);
     readln
end.
```

Pentru cazul numerelor despărţite prin virgule se declară în locul variabilei reale *r* o variabilă *s* string (dacă liniile fişierului nu pot fi mai lungi de 255 de caractere) şi se înlocuiesc liniile comentate ({*}) cu următoarele:

```
i:=1;
readln(f,s);
   for j:=1 to length(s) do
      if s[j]=',' then i:=i+1;
if s<>'' then
   writeln('Nr. de numere aflate pe linia ',k,':',i)
else writeln('Nr. de numere aflate pe linia ',k,':',0)
```

Observaţii:

– în niciunul din cazuri nu s-a luat în consideraţie corectitudinea numerelor de pe o linie şi nici nu s-a prevăzut cazul existenţei mai multor virgule între două numere;

– dacă s-ar dori numărarea cuvintelor de pe o linie, programul nu ar mai fi atât de simplu deoarece interpretarea unui şir de caractere trebuie făcută până la dimensiunea sa maximă sau până la EOLN, spaţiile considerându-se caractere oarecare din şir;

– cu precizarea că funcţia *readln* înaintează în fişier până după EOLN, urmăriţi ce se întâmplă dacă ultima linie din fişier este albă şi găsiţi o soluţie şi pentru acest caz particular.

Scanned with OKEN Scanner

7.4 (IX)

Scrieţi un program care transformă toate literele mari dintr-un fişier dat în litere mici. Încercaţi apoi să păstraţi literele neschimbate numai în şirurile de caractere aflate între ghilimele.

R 7.4

Fişierele text existente pot fi deschise numai pentru citire (deschiderea pentru scriere realizată cu procedura *rewrite* pentru un fişier existent anterior conduce la pierderea informaţiei din fişier şi are sens numai atunci când se creează fişierul); deci nu se poate scrie în fişierul dat. Atunci se creează un alt fişier text de lucru, în care sunt copiate unul câte unul caracterele din fişierul dat, după ce au fost convertite la litere mici. După închiderea celor două fişiere, fişierul dat se şterge iar numele lui este folosit la redenumirea fişierului de lucru.

```
program lit_mari_mici;
var nume:string;
    c:char;
    f,g:text;
begin
     write('Numele fisierului:');readln(nume);
     assign(f,nume);assign(g,'manev');
     reset(f);rewrite(g);
     while not eof(f) do
           begin
                read(f,c);
{*}             if c in ['A'..'Z'] then c:=chr(ord(c)+32);
                write(g,c);
     end;
     close(f);
     close(g);
     erase(f);
     rename(g,nume)
end.
```

Pentru a doua cerinţă folosim o variabilă *citat* care este iniţial false şi care-şi comută valoarea cu fiecare caracter " întâlnit. Astfel liniile {*} se schimbă în felul următor:

```
if c='"' then citat:=not citat;
if (not citat)and(c in['A'..'Z']) then
  c:=chr(ord(c)+32)
```

Observaţie: Dacă dorim să lăsăm neschimbate caracterele cuprinse între apostrofuri, înlocuim `if c ='"'...` cu `if c = ''''...` (în constantele string, caracterul ' se scrie dublu pentru a-l deosebi de separatorul ').

7.5 (IX)

Fişierul text **fiş.dat** conţine numere şi cuvinte pe mai multe linii, fiecare linie având maximum 255 de caractere. Să se realizeze un program care inventariază conţinutul fişierului, afişând pentru fiecare linie şi pentru întregul fişier numărul de caractere, numărul de cuvinte şi numărul de valori numerice.

Scanned with OKEN Scanner

R 7.5

Nu putem analiza cuvintele ca sens, dar putem încerca să transformăm fiecare „cuvânt" în număr real şi, în funcţie de reuşită (valoarea variabilei cod ce întoarce codul de eroare în urma aplicării procedurii *val*) decidem dacă este număr sau un cuvânt oarecare. Mulţimea *car* conţine acele caractere ce pot apărea în cuvinte şi numere, celelalte considerându-se separatori. Înregistrarea *lin* de tip *valori* conţine caracteristicile unei linii (lungime, număr de valori numerice, număr de cuvinte), iar înregistrarea *tot* conţine caracteristicile întregului fişier. Înregistrarea constantă *zero* serveşte la iniţializarea înregistrărilor *lin* şi *tot*.

```
program compozitie_fisier;
const car=['A'..'Z','a'..'z','0'..'9','.']; {Caractere propriu-zise}
type valori= record
                    lung:longint;
                    nr,cuv:word
             end;
const zero:valori=(lung:0;nr:0;cuv:0);
var f:text;
    sir:string;
    lin,tot:valori;
    i,cod,poz:integer; n:real;
begin
     assign(f,'fis.dat');
     {$I-} reset(f); {$I+}
     if ioresult<>0 then begin
        writeln('Nu exista fisierul fis.dat');
        halt
     end;
     tot:=zero; i:=0;
     writeln('Linie  Caractere  Cuvinte   Numere ');
     while not eof(f) do
       with lin do begin
         lin:=zero;
         readln(f,sir);
         lung:=length(sir);
         inc(i);
         while sir<>'' do begin
           poz:=0;
           while (poz<lung)and(sir[poz+1] in car) do poz:=poz+1;
           if poz<>0 then begin
              val(copy(sir,1,poz),n,cod);
              if cod=0 then inc(nr)
                       else inc(cuv)
           end;
           delete(sir,1,poz+1)
         end;
         writeln(i:4,lung:10,cuv:10,nr:10);
         inc(tot.lung,lung);
         inc(tot.cuv,cuv);inc(tot.nr,nr)
       end;
       writeln('Total',tot.lung:9,tot.cuv:10,tot.nr:10);
       close(f)
end.
```


Scanned with OKEN Scanner

7.6 (VIII)

În fişierul text 'masini.old' există mai multe linii, pe fiecare linie aflându-se numerele de înmatriculare de tip vechi ale unor autoturisme. Numerele vechi sunt de forma *nr-judeţ-număr*, cu *nr* de minim una şi maxim două cifre, *judeţ* de două litere (excepţie Bucureştiul cu litera B) şi *număr* de minim trei şi maxim cinci cifre. Se vor afişa pe rând numerele vechi şi se va solicita codul de trei litere solicitat, cod specific numerelor de înmatriculare de tip nou. Dacă este posibil, se va atribui maşinii noul cod împreună cu numărul de două cifre obţinut în funcţie de numărul de maşini cu acelaşi judeţ şi acelaşi cod întâlnite anterior. Se vor trece noile coduri în fişierul 'masini.nou', iar numerele maşinilor care nu respectă formatul vor fi afişate pe ecran câte unul pe linie, cu specificaţia „maşini străine".

Exemplu: Dacă fişierul 'masini.old' conţine liniile

12-B-1970
15-PH-211
WW 6483.X
89-TM-15623
3-B-12903
63-B-2407

se vor solicita coduri noi pentru numerele de pe liniile 1, 2, 4, 5 şi 6; dacă se furnizează codurile ABB,MNX,EDI,SAL,ABB, fişierul 'masini.nou' va conţine liniile:

B-01-ABB
PH-01-MNX
TM-01-EDI
B-01-SAL
B-02-ABB

iar pe ecran apare: **`WW 6483.X masini straine.`**

R 7.6

Codurile se transferă modificate din fişierul 'masini.old' în fişierul 'masini.nou' sau 'masini.str', pentru numerele vechi valide calculând noul număr (care poate fi cel mult 99) cu ajutorul unui vector de înregistrări.

```
program masini;
type masina=record
                  cod:string[5]; {cod+judet}
                  nr:0..99
            end;
var vechi,nou,st:text;
    m:array[1..9000]of masina;
    ant:byte;codd:string[3];s,s_nou:string[30];
function valid(x:string):boolean;
var cod_err,l:integer;
begin
     valid:=false;
     val(copy(x,1,pos('-',x)-1),l,cod_err);
     if (cod_err<>0)or(l>99) then exit;
     delete(x,1,pos('-',x));
     s_nou:=copy(x,1,pos('-',x)-1);
     l:=length(s_nou);
```


Scanned with OKEN Scanner

```
      if not(l in [1..2])or(l=1)and(s_nou<>'B') then exit;
      delete(x,1,pos('-',x));
      if not(length(x)in[3..5]) then exit;
      val(x,l,cod_err);
      if cod_err<>0 then exit;
      valid:=true
end;
function posibil:boolean;
var i:integer;n:string;
begin
      posibil:=false;
      i:=1;
      m[ant+1].cod:=codd+s_nou;
      while m[i].cod<>codd+s_nou do inc(i);
      if i=ant+1 then begin
         inc(ant); m[ant].nr:=1
      end
      else
          if m[i].nr=99 then exit else inc(m[i].nr);
      str(m[i].nr,n);
      if m[i].nr<10 then n:='0'+n;
      s_nou:=s_nou+'-'+n+'-'+codd;
      posibil:=true
end;
begin
      fillchar(m,sizeof(m),0);
      assign(vechi,'masini.old'); reset(vechi);
      assign(nou,'masini.nou'); rewrite(nou);
      assign(st,'masini.str'); rewrite(st);
      ant:=0;
      while not seekeof(vechi) do begin
            readln(vechi,s);
            if valid(s) then begin
               writeln(s:20);
               repeat
                     write('Noul cod (3 litere mari):'); readln(codd)
               until posibil;
               writeln(nou,s_nou);
            end
            else writeln(st,s)
      end;
      close(vechi);close(nou);
      close(st);reset(st);
      while not seekeof(st) do begin
            readln(st,s);
            writeln(' ':20-length(s),s,'  masina straina')
      end;
      close(st)
end.
```

Observaţie: Codurile deja existente ţinute în memorie, pot depăşi numărul 9000, supradimensionarea vectorului m depăşind segmentul de date de 64K (eroare de alocare a memoriei, semnalată la compilare); soluţia este fie să se lucreze cu lista maşinilor alocată dinamic, fie să se formeze un fişier cu valorile lui m care, dacă este fişier text, va fi tot timpul închis şi redeschis pentru fiecare maşină căutată (activitate de uzură pentru disc), deci e preferabil un fişier cu tip (accesul fiind direct).

Scanned with OKEN Scanner

7.7 (IX)

Să se creeze un fişier cu tip care cuprinde un număr oarecare de numere întregi aleatoare de cel mult patru cifre fiecare. Să se afişeze prima secvenţă, maximală ca lungime, de numere pare din fişier.

R 7.7

Căutarea secvenţei de lungime maximă implică:

a) parcurgerea secvenţială a fişierului;

b) revenirea la poziţia memorată ca început al secvenţei;

c) scrierea elementelor din secvenţă.

```
program secventa_max_para;
type sir=file of integer;
const max=50;
var
   f:sir; sec:boolean;
   n,i,lung,poz,pmax,lmax:integer;
begin
     assign(f,'f1.dat');rewrite(f);
     randomize;
     for i:=1 to random(max) do begin
         n:=random(10000);
         write(f,n);
     end;
     writeln('S-au generat numerele:');
     seek(f,0);                    {pozitionare inceput fisier}
     poz:=0;lung:=0;
     lmax:=lung;pmax:=poz;
     while not eof(f) do begin
           read(f,n);write(n:5);
           if n mod 2 =0 then lung:=lung+1
           else begin
               if lung>lmax then begin
                  lmax:=lung;
                  pmax:=poz
               end;
               poz:=filepos(f);
               lung:=0
           end
     end;
     if lung>lmax then begin
       lmax:=lung;
       pmax:=poz
     end;
     seek(f,pmax);
     writeln;
     for i:=1 to lmax do begin
         read(f,n);
         write(n:5)
     end;
     writeln;
     close(f)
end.
```


Scanned with OKEN Scanner

7.8 (IX)

Să se creeze o nouă agendă sau să se afişeze conţinutul celei vechi (existând şi opţiunea de a crea o nouă agendă în locul celei vechi) oferind posibilitatea consultării agendei în vederea obţinerii datelor referitoare la o anumită persoană.

R 7.8

Programul propus dă o rezolvare de principiu a problemei, amănuntele „de design" ale utilizării fiind omise (ele rămân în sarcina celui care-şi propune să realizeze o agendă cu adevărat utilă):

```
program agenda;
uses crt;
type inreg=record
                nume:string[20];
                adresa:string;
                telefon:string[20]
           end;
var f:file of inreg;
    inr:inreg;
procedure creare;
begin
     rewrite(f);
     write('Nume:');
     readln(inr.nume);
     while inr.nume<>'' do begin
           write('Adresa:');
           readln(inr.adresa);
           write('Telefon (tara-jud-telef/interior):');
           readln(inr.telefon);
           write(f,inr);
           write('Nume (ENTER pt. iesire):');
           readln(inr.nume)
     end;
     writeln('fisierul are',filesize(f),'inregistrari');
     close(f)
end;
procedure listare;
var i:word;
begin
     {$I-}reset(f){$I+};
     if ioresult<>0 then writeln('Fiserul nu a fost creat')
     else
         for i:=1 to filesize(f) do
             with inr do begin
                  read(f,inr);
                  writeln(nume,'*',adresa,'*',telefon);
                  if wherey=25 then
                     begin
                          readkey;
                          clrscr
                     end
             end;
     close(f)
end;
```


Scanned with OKEN Scanner

```
procedure cautare;
var num:string;
    este:boolean;
begin
     reset(f);  este:=false;
     write('Numele cautat:'); readln(num);
     while not eof(f) do
          with inr do begin
               read(f,inr);
               if nume=num then begin
                  writeln(nume,'*',adresa,'*',telefon);
                  if wherey=25 then begin readkey;clrscr end
                  este:=true
               end
          end;
     if not este then writeln('Numele cautat nu exista');
     close(f);
end;
begin
     assign(f,'AGENDA.DAT'); {$I-}reset(f){$I+};
     if ioresult=0 then begin
        writeln('Fisierul agenda.dat exista deja'); listare;
        writeln('Creati altul ? (D/N) ');
        if upcase(readkey)='D' then creare
     end
     else creare;
     clrscr;
     cautare
end.
```

7.9 (IX)

Să se creeze un fişier cu tip care conţine informaţii referitoare la pacienţii unui laborator de spital: nume, vârstă, diagnostic şi adresă (din spital, dacă este spitalizat, de la domiciliu altfel).

R 7.9

Se creează un fişier cu înregistrări variabile în care, în funcţie de statutul pacientului (spitalizat sau nu) se definesc câmpurile *sect* şi *sal* respectiv *adr* care este la rândul lui o înregistrare:

```
program spitalizare;
type inreg=record
                  nume :string[20];
                  varsta:byte;
                  boala:string[20];
                  case spit:boolean of
                       true:(sect:string[15];     {sectie}
                             sal:byte);           {salon}
                       false:(adr:record
                                    str:string[15];
                                    nr:byte;
                                    ap:word
                              end;)
          end;
```


Scanned with OKEN Scanner

```
var f:file of inreg;b:inreg;
    c:char;i,n:byte;
begin
     assign(f,'inreg.dat');
     rewrite(f);readln(n);
     for i:=1 to n do
         with b do begin
              write('nume :');readln(nume);
              write('varsta: ');readln(varsta);
              write('boala:');readln(boala);
              write('Este spitalizat (D/N)? ');readln(c);
              spit:=boolean(upcase(c)='D');
              if spit then begin
                 write('sectia:');readln(sect);
                 write('salon:');readln(sal);
              end
              else begin
                   write('strada ');readln(adr.str);
                   write('nr ');readln(adr.nr);
                   write('apartament ');readln(adr.ap)
              end;
              write(f,b)
         end;
     close(f)
end.
```

Propunem ca exerciţiu completarea programului cu rutine de listare şi căutare pentru acest fişier.

7.10 (IX)

Să se ordoneze descrescător după medii, înregistrările unui fişier care conţine datele şi rezultatele candidaţilor la un examen de admitere în clasa a IX-a. Fiecare înregistrare este compusă din cinci câmpuri: un nume (variabilă tip string cu 20 caractere), trei numere reale reprezentând notele la cele două probe şi media de examen obţinută, şi un câmp logic ce memorează rezultatul final (ADMIS sau RESPINS). Programul va afişa rezultatul examenului de admitere la liceu în funcţie de un număr citit de la tastatură reprezentând numărul de locuri disponibile.

Exemplu:

Fişierul cu înregistrările

```
Popescu Dan,6.33,9.27
Ilie Marina,5,7.50
Dragu Georgeta,9,10
Ion Dorin,9,9.24
```

pentru numărul de locuri n_loc=3 va fi restructurat conţinând următoarele înregistrări complete:

```
Dragu Georgeta,9,10,9.50,true
Ion Dorin,9,9.24,9.12,true
Popescu Dan,6.33,9.27,7.80,true
Ilie Marina,5,7.50,6.25,false
```


Scanned with OKEN Scanner

R 7.10

Prezentăm un algoritm de ordonare în memorie:

```
program examen;
uses crt;
type concurent=record
                    nume:string[20];
                    rom,mat,med_adm:real;
                    admis:boolean
               end;
     fisier=file of concurent;
     med_mem=record
                  med:real;
                  nr_art:integer
             end;
var f,f_rez:fisier;
    c:concurent; n,i:integer;
procedure creare;
begin
     rewrite(f);
     write('Cati candidati sunt?'); readln(n);
     for i:=1 to n do
         with c do begin
              write('Numele candidatului ',i:3,': ');readln(nume);
              write('Notele (romana matematica) :');readln(rom,mat);
              med_adm:=trunc((rom+mat)/2*100)/100;
              write(f,c)
        end;
     close(f)
end;
procedure listare(var f:fisier);
begin
     reset(f);clrscr;
     for i:=1 to filesize(f) do
         with c do begin
              read(f,c);write(i:3,' ',nume,' ':25-length(nume),
                              rom:8:2,mat:8:2,med_adm:8:2);
              if admis then writeln(' ADMIS')
              else writeln(' RESPINS');
              if wherey=25 then begin readkey;clrscr end
         end;
     close(f)
end;
procedure ordonare;
var ordo:boolean;
    aux:med_mem;
    m:array [0..999] of med_mem;
    n_loc:integer;
begin
     reset(f); n:=filesize(f);
     for i:=0 to n-1 do begin
         read(f,c);
         m[i].med:=c.med_adm;
         m[i].nr_art:=i
     end;
```


Scanned with OKEN Scanner

```
      repeat
            ordo:=true;
            for i:=0 to n-2 do
                if m[i].med < m[i+1].med then begin
                   aux:=m[i];
                   m[i]:=m[i+1];
                   m[i+1]:=aux;
                   ordo:=false
            end
      until ordo;
      write('Introduceti numarul de locuri : ');
      readln(n_loc);
      rewrite(f_rez);
      for i:=0 to n-1 do
          with c do begin
               seek(f,m[i].nr_art);
               read(f,c);
               if (rom>=5)and(mat>=5)and(n_loc>0) then begin
                  admis:=true;
                  dec(n_loc)
               end
               else admis:=false;
               write(f_rez,c)
          end;
      close(f);
      close(f_rez);
      listare(f_rez);
      writeln('Locuri neocupate:',n_loc);
end;
begin
   assign(f,'examene.dat');
   assign(f_rez,'rezultat.dat');
   {$I-}reset(f){$I+};
   if ioresult<>0 then creare
                  else listare(f);
   writeln('Doriti ordonarea candidatilor ? (D/N) ');
   if upcase(readkey)='D' then ordonare;
   {$I-}reset(f_rez){$I+};
   if ioresult<>0 then listare(f_rez)
end.
```

De cele mai multe ori, numărul de înregistrări este foarte mare, nefiind posibilă memorarea tuturor mediilor. De aceea se preferă ordonarea pe disc, deşi aceasta distruge fişierul de date anulând mediile concurenţilor care au fost deja transferaţi în fişierul de rezultate:

```
procedure ordonare;
var p_max,n_loc:integer;
    med_max:real;
begin
     reset(f);
     rewrite(f_rez);
     write('Introduceti numarul de locuri : ');
     readln(n_loc);
     n:=filesize(f);
```


Scanned with OKEN Scanner

```
    for i:=1 to n do
        with c do begin
             med_max:=0; seek(f,0);
             for j:=1 to n do begin
                 read(f,c);
                 if med_adm>med_max then begin
                    med_max:=med_adm;
                    p_max:=filepos(f)-1
                 end
             end;
             seek(f,p_max); read(f,c);
             if (rom>=5)and(mat>=5)and(n_loc>0) then begin
                admis:=true; dec(n_loc)
             end
             else admis:=false;
             write(f_rez,c);
             med_adm:=0;
             seek(f,p_max);write(f,c)
        end;
    close(f);
    close(f_rez);
    listare(f_rez);
    writeln('Locuri neocupate:',n_loc)
end;
```

Vă mai recomandăm spre studiu metoda de ordonare cu interschimbare chiar în fişier, propusă în [15].

7.11 (IX)

Fişierul 'produse.dat' conţine informaţii despre produsele dintr-un magazin alimentar: numele produsului (de maxim 20 de caractere), preţul unitar (număr întreg, maxim 100000), cantitatea (număr real) şi termenul de garanţie (sub forma *zi–luna–an* când expiră termenul de folosinţă al produsului). Să se afişeze în funcţie de dată (şi să se elimine din fişier) produsele expirate şi costul lor total.

R 7.11

Propunem o modalitate mai eficientă de eliminare, faţă de cea în care rescriem într-un alt fişier toate înregistrările, mai puţin cea eliminată: se rescrie ultima înregistrare peste înregistrarea de eliminat şi se trunchiază fişierul exact în dreptul ultimei înregistrări (care astfel dispare). Bineînţeles această metodă poate fi acceptată numai dacă nu ţinem la ordinea iniţială din fişier.

```
program produse;
uses crt,dos;
type produs=record
               nume:string[20];
               pret:longint;
               cant:real;
               gar:record
                        zi,luna:byte;
                        an:word
                   end
          end;
```


Scanned with OKEN Scanner

```
var f:file of produs;
    p:produs;
procedure creare;
var n,i:word;
begin
     rewrite(f);
     write('Cate produse se introduc ?');readln(n);
     for i:=1 to n do
         with p,gar do begin
              write('Numele:');readln(p.nume);
              write('Pret/unitate Cantitate :'); readln(pret,cant);
              write('Data expirare:'); readln(zi,luna,an);
              write(f,p);
         end;
     close(f)
end;
procedure listare;
var i:word;
begin
     {$I-}reset(f){$I+};
     if ioresult<>0 then writeln('Fisierul nu a fost creat')
     else
         for i:=1 to filesize(f) do
             with p,gar do begin
                  read(f,p);
                  writeln(i:3,nume:20,pret:7,cant:10:2,zi:2,'-',
                          luna:2,'-',an:4);
                  if wherey=25 then begin readkey;clrscr end
             end;
     close(f)
end;
procedure actualizare;
var z,l,a,s:word;paguba:real;
begin
     getdate(a,l,z,s);
     writeln(z:3,l:3,a:3);
     reset(f);paguba:=0;
     while not eof(f) do
           with p,gar do begin
                read(f,p);
                if (an<a)or(an=a)and(luna<l)or(an=a)and
                   (luna=l) and (zi<z) then begin
                    writeln(nume:20,pret:7,cant:10:2,zi:2,
                            '-',luna:2,'-',an:4);
                    paguba:=paguba+pret*cant;
                    s:=filepos(f)-1; {eliminarea}
                    seek(f,filesize(f)-1); read(f,p);
                    seek(f,s); write(f,p);
                    seek(f,filesize(f)-1);
                    truncate(f)
                end
                end;
     writeln('Costul produselor expirate:',paguba:15:2);
     close(f);readln
end;
```


Scanned with OKEN Scanner

```
begin
     clrscr;
     assign(f,'produse.dat');
     {$I-}reset(f){$I+};
     if ioresult=0 then begin
        writeln('Fisierul exista deja');
        listare;
        writeln('Creati altul ? (D/N) ');
        if upcase(readkey)='D' then creare
     end
     else creare;
     clrscr;
     actualizare;
     listare;
     readln
end.
```

7.12 (IX)

Fişierul *persoane.dat* conţine numele, adresele (pe câte 40 de caractere) şi data naşterii (pe 4 octeţi) ale unui număr nelimitat de persoane. Să se creeze 12 fişiere text, fiecare fişier conţinând numele şi adresele persoanelor dintr-o aceeaşi zodie.

R 7.12

Algoritmul transferă informaţie între un fişier cu tip şi mai multe fişiere text, organizate într-un tablou de fişiere. Vă recomandăm să urmăriţi modul de lucru în ecran realizat de procedurile *macheta* şi *creare*, care vă pot fi utile şi pentru alte tipuri de aplicaţii:

```
program zodiac;
uses crt;
const z:array[1..12]of string[9] = ('CAPRICORN','VARSATOR','PESTI',
                                    'BERBEC', 'TAUR','GEMENI','RAC',
                                    'LEU','FECIOARA', 'BALANTA',
                                    'SCORPION','SAGETATOR');
      l:array[1..12]of byte=(31,28,31,30,31,30,31,31,30,31,30,31);
      zz:array[1..12]of word= (19,51,79,110,140,171,201,232,263,293,
                               324,354);
        {numar zile trecute corespunzator fiecarui inceput de zodie}
type
     sir=string[40];
     data= record
                 zi:1..31;
                 lu:1..12;
                 an:word
     end;
     pers= record
                nume,adresa:sir;
                data_n:data
           end;
var f:file of pers;
     p:pers; i,zile:word;
     fz:array[1..12]of text;
```


Scanned with OKEN Scanner

```
procedure macheta(a,b:string); {a-titlul ferestrei, b-mesaj}
begin
     textbackground(red);textcolor(white);
     window(1,1,80,25);clrscr; write(a);
     window(15,10,65,16);textbackground(black);
     writeln(b,#13#10);
     writeln('Numele    :'); writeln('Adresa    :');
     writeln('Anul_nast:'); writeln('Luna_nast:');
     write('Ziua_nast:');
     window(25,12,65,16)
end;
procedure creare;
var s:sir;er:integer;
  function iese(var r:sir):boolean;
  var c:char;
  begin
       r:='';iese:=true;
       repeat
            while keypressed do readkey;
            c:=upcase(readkey);
            case c of
                 #27: exit;
                 #8 : delete(r,length(r),1);
                 #32..#96:r:=r+c
            end;
            delline;
            write(#13,r) {de la capatul aceluiasi rand}
       until (c=#13)or(length(r)=40);
       writeln(#7);iese:=false
  end;
begin
     rewrite(f);
     macheta('     CREAREA FISIERULUI DE DATE PERSOANE.DAT',
             '        Pentru iesire apasati ESC  ');
     repeat
          clrscr;
          with p,data_n do begin
               clrscr;
               if iese(nume) or iese(adresa) then exit;
               repeat
                    gotoxy(1,3);delline; if iese(s) then exit;
                    val(s,an,er)
               until er=0;
               repeat
                    gotoxy(1,4);delline; if iese(s) then exit;
                    val(s,lu,er)
               until (er=0)and(lu in [1..12]);
               repeat
                    gotoxy(1,5);delline; if iese(s) then exit;
                    val(s,zi,er)
               until (er=0)and(zi in [1..l[lu]])
          end;
          write(f,p)
     until false
end;
```


Scanned with OKEN Scanner

```
procedure listare;
begin
     macheta('  CONTINUTUL FISIERULUI DE DATE PERSOANE.DAT',
                '     Apasati ENTER sau ESC     ');
     seek(f,0);
     for i:=1 to filesize(f) do
         with p,data_n do begin
              read(f,p);clrscr;
              writeln(nume);writeln(adresa);
              write(an,#13#10,lu,#13#10,zi);
                   if readkey=#27 then exit
         end;
end;
begin
     assign(f,'PERSOANE.DAT');
     {$I-}reset(f);{$I+}
     if ioresult<>0 then creare
                    else listare;
     window(20,20,60,25);clrscr;
     write('Creati alt fisier ? (D/N)');
     if upcase(readkey)='D' then creare;
     for i:=1 to 12 do begin
         assign(fz[i],z[i]+'.dat');
         rewrite(fz[i]);
         close(fz[i])
     end;
     seek(f,0);
     while not eof(f) do
           with p do begin
                read(f,p);
                zile:=0;
                for i:=1 to data_n.lu-1 do zile:=zile+l[i];
                zile:=zile+data_n.zi;
                i:=1;
                while (i<=12)and(zile>zz[i]) do i:=i+1;
                if i=13 then i:=1;
                append(fz[i]);
                writeln(fz[i],nume,' ',adresa);
                close(fz[i])
           end;
     close(f);
     window(1,1,80,25);clrscr;
     writeln('Studiati continutul fisierelor nume_zodie.dat');
     readln
end.
```

7.13 (IX)

Să se scrie un program de testare şi exersare a abilităţii de folosire a tastaturii alfanumerice. Redactaţi toate procedurile şi funcţiile într-o unitate numită *tasta*.

R 7.13

Unitul *tasta* deserveşte programul principal *tastatura*. Procedurile *win* activează una dintre cele 4 ferestre utile, procedura *test* afişează succesiv câte 1, 2, ... caractere (pe măsură ce utilizatorul dobândeşte dexteritate), aşteptând un

Scanned with OKEN Scanner

timp dat (proporţional cu numărul de caractere cerute) să fie apăsate tastele corespunzătoare. Procedura *timp* foloseşte unitatea DOS pentru a calcula timpul scurs de la afişarea secvenţei de caractere. Se utilizează fişierul de rezultate 'elevi.dat', actualizat după fiecare test prin procedura *scor* şi afişat prin procedura *rezultate*.

```
unit tasta;
interface
uses crt,dos;
type tempo=record
                h,m,s,ss:word;
           end;
     elev=record
                num:string[40];
                cls:string[3];
                niv:byte;
                tmp:tempo;
                nota:byte;
          end;
const
     gres_max=20;
     t_max=5;
var
   timp_ini,timp:tempo;
   gres:word;
   nivel:byte;
   ra:integer;
   e:elev;
procedure win(i:byte;s:string);
procedure nume;
procedure rezultate;
function v_timp:boolean;
procedure test;
procedure scor;

implementation
procedure win;
var
   k:integer;
begin
    case i of
      1:begin{fereastra meniu}
              window(1,1,80,1);textbackground(blue);clrscr;
              gotoxy(10,1);textcolor(white);write('R');
              textcolor(black);write('ezultate');
              gotoxy(34,1);textcolor(white);write('T');
              textcolor(black);write('est');
              gotoxy(57,1);textcolor(white);write('S');
              textcolor(black);write('fârsit');
        end;
      2:begin
              win(3,'');clrscr;
              textcolor(white);
              write(s)
        end;
```


Scanned with OKEN Scanner

```
      3:begin{fereastra-dialog-only}
              window(1,25,80,25);
              textbackground(blue);
              textcolor(white);
        end;
      4:begin
          window(25,6,55,8);textbackground(blue);clrscr;
              gotoxy(6,2);textbackground(black);
              textcolor(white);write(s);
        end;
      5:begin
              window(25,10,55,12);textbackground(blue);clrscr;
              gotoxy(2,2);textbackground(black);
              textcolor(white);write(s);
        end;
    end;
end;
function v_timp;
var
   t:tempo;
   rest:integer;
begin
     with t do gettime(h,m,s,ss);
     v_timp:=false;
    rest:=t_max+nivel div 2-(t.h-timp.h)*3600-
          (t.m-timp.m)*60-(t.s-timp.s);
     win(3,'');
     write('Timp: 00:',rest div 60,':',rest mod 60);
     if rest<=0 then v_timp:=true;
end;
procedure test;
var
   i,j:integer;
   ch:char;
   sir_dat,rasp:string;
begin
     repeat
        for i:=1 to 20-nivel do begin
            sir_dat:='';
            randomize;
            for j:=1 to nivel+1 do
                sir_dat:=sir_dat+chr(32+random(95));
            win(4,sir_dat);
            win(5,'Astept!');
            rasp:='';
            with timp do gettime(h,m,s,ss);
            while keypressed do readkey;
            repeat
                 repeat until keypressed or v_timp;
                 if not(v_timp) then begin
                    ch:=readkey;
                    if ord(ch)>31 then rasp:=rasp+ch;
                 end;
            until v_timp or(length(rasp)=(nivel+1))or(ord(ch)<32);
```


Scanned with OKEN Scanner

```
            if ch=#27 then exit;
            sound(300);delay(300);nosound;
            if rasp<>sir_dat then begin
               inc(gres);
               if gres>gres_max then exit;
               win(3,''); gotoxy(20+gres,1); write(' ');
               win(5,'Ai tastat:'+rasp);
               for j:=256 downto 150 do begin
                   sound(j*2);delay(30);nosound;
               end;
            end;
         end;
         nivel:=nivel+1;
      until false;
end;

procedure nume;
begin
     win(2,#7);
     write('  Numele  : ');readln(e.num);
     write('  Clasa  : ');readln(e.cls);
     win(2,'                                    '); {alt+179}
     with timp_ini do gettime(h,m,s,ss);
     gres:=0;nivel:=0;
end;

procedure scor;
var
   f:file of elev;
   t:tempo;
   calc,s,i:longint;
begin
     assign(f,'elevi.dat');
     {$I-}reset(f){$I+};
     if ioresult<>0 then rewrite(f);
     e.niv:=nivel;
     with t do gettime(h,m,s,ss);
     calc:=(t.h-timp_ini.h)*3600+(t.m-timp_ini.m)*60+
           (t.s-timp_ini.s);
     e.tmp.h:=calc div 3600;
     e.tmp.m:=calc div 60;
     e.tmp.s:=calc div 60-e.tmp.h*60;
     e.nota:=round(10-(5-nivel)*2);
     if e.nota>10 then e.nota:=10;
     seek(f,filesize(f));
     write(f,e);
     close(f);
end;

procedure rezultate;
var
   parola:string;
   f:file of elev;
begin
     assign(f,'elevi.dat');
     {$i-}reset(f);{$i+}
```


Scanned with OKEN Scanner

```
     if ioresult<>0 then rewrite(f);
     window(1,2,80,24);
     textbackground(black);
     while not(eof(f)) do begin
           read(f,e);
           with e do writeln(num:40,cls:5,' ',niv:3,' ',tmp.h:2,
                     ':',tmp.m:2,':',tmp.s:2,nota:10);
     end;
     close(f);
     write('Apasati tasta ENTER ...');
     readln;
end;
end.

program tastatura;
uses crt,tasta;
var
   ch:char;
begin
     repeat
       textbackground(black);
       clrscr;win(1,' ');
       win(2,'Apasa tasta corespunzatoare din meniu!');
       repeat
         repeat until keypressed;
         ch:=upcase(readkey);
       until ch in['R','T','S'];
       case ch of
            'R':rezultate;
            'T':begin
                   nume; test; scor;
                   window(1,2,80,24);
                   clrscr;
                end;
       end;
     until ch='S';
     textbackground(black);
     textcolor (white);clrscr
end.
```

7.14 (IX)

Să se numere frecvenţa apariţiei unui caracter (dat prin codul său ASCII) într-un fişier oarecare precizat.

R 7.14

Se deschide fişierul ca fişier cu tip folosind un bloc de lungime determinată, astfel încât să se obţină conţinutul fişierului printr-un număr întreg de citiri:

```
program aparitii;
var fisier:file;
    bloc:array[1..10000]of char;
    a:char;
    nume:string;
    i,j,na:word;
```


Scanned with OKEN Scanner

```
begin
     write('Codul caracterului:');
     readln(i);a:=chr(i);
     write('Numele fisierului:');
     readln(nume);
     assign(fisier,nume);
     {$I-}reset(fisier,1);{$I+};
     if ioresult<>0 then writeln('Fisierul nu exista')
     else begin
          i:=10000;
          na:=0;
          if filesize(fisier)<i then i:=filesize(fisier);
          while filesize(fisier) mod i <>0 do dec(i);
          close(fisier);
          reset(fisier,i);
          while not eof(fisier) do begin
                blockread(fisier,bloc,1);
                for j:=1 to i do
                    if bloc[j]=a then inc(na)
          end
     end;
     writeln('Numarul de aparitii ale lui #',ord(a),' este:',na);
     close(fisier)
end.
```

7.15 (IX)

Se citesc numele a două fişiere (acelea care nu sunt în directorul curent se precizează împreună cu calea absolută sau relativă). Să se decidă dacă fişierele sunt identice sau nu, şi în cazul detectării unei diferenţe, se va afişa poziţia unde a apărut diferenţa şi în ce constă aceasta.

R 7.15

Se deschid ambele fişiere ca fişiere fără tip, deoarece nu sunt create în format standard (Pascal) de fişiere cu tip şi nici fişiere text nu pot fi. Amintim că pentru fişierele text, detectarea sfârşitului de fişier se face prin depistarea caracterului ^Z, în fişierul nostru putând exista mai multe astfel de caractere pe care programul le-ar desemna ca sfârşit de fişier (de fapt doar pe primul, deoarece pe celelalte nu mai apucă să le citească).

Programul dă un preaviz că fişierele nu sunt identice, dacă observă că acestea au lungimi diferite, sau scanează fişierul şi se opreşte pe prima diferenţă. S-au folosit înregistrări de un octet pentru a găsi prin citire directă (fără alte operaţii) octetul diferit, deşi ar fi fost mai eficientă citirea pe blocuri mari. Scanarea se poate întrerupe prin apăsarea oricărei taste.

```
program compar;
uses crt;
var f,g:file;
    a,b:byte;
    n1,n2,i:integer;
    num1,num2:string;
    la_fel:boolean;
```


Scanned with OKEN Scanner

```
begin
     write('Numele unui fisier:');readln(num1);
     write('Numele celuilalt  :');readln(num2);
     assign(f,num1);assign(g,num2);
     reset(f,1);reset(g,1);
     if filesize(f)<>filesize(g) then
        writeln('Fisierele au lungimi diferite');
     la_fel:=true;
     i:=1;
     while la_fel and not eof(f) do begin
           blockread(f,a,1,n1); blockread(g,b,1,n2);
           write(#13'Octetul ',i,': ',a:6);
           if a<>b then la_fel:=false;
           if keypressed then begin
              writeln('  Intrerupere!'); la_fel:=false
           end;
           i:=i+1
     end;
     if not(eof(f) and eof(g)) then
        writeln(#13#10'Octetul ',i-1,': ',b:6)
     else writeln(#10#13'Fisierele sunt identice');
     close(f);close(g)
end.
```

7.16 (IX)

Să se afişeze fişierele din directorul curent, ştergând selectiv pe cele pentru care optează utilizatorul.

R 7.16

Se folosesc procedurile *findfirst* şi *findnext* cu care se poate naviga printre fişierele specificate de primul parametru al lui *findfile* (în cazul nostru '*.*' reprezintă toate fişierele din directorul curent) selectând primul respectiv următorul fişier cu atributele specificate prin al doilea parametru al lui *findfile*. Variabila *doserror* întoarce un număr reprezentând codul de eroare raportat de sistemul de operare; în contextul problemei, *doserror* < > 0 când nu se mai execută corect trecerea la următoarea intrare din director (nu mai există nici un fişier). Tipul *searchrec* este un tip înregistrare care conţine pe lângă câmpul *name* (string[12]) şi alte informaţii aferente fişierului (dată, lungime, etc).

```
program sterge;
uses dos,crt;
var s:searchrec;ch:char;f:file;
begin
     ch:=' '; findfirst('*.*',anyfile,s);
     while (doserror=0)and(ch<>#27) do begin
           writeln(s.name,' ':20-length(s.name),'Il stergi? (Y/N)');
           ch:=upcase(readkey);
           if ch='Y' then begin
              assign(f,s.name);erase(f);
              sound(200);delay(200);nosound
           end;
           findnext(s)
     end
end.
```


Scanned with OKEN Scanner

Probleme propuse

7.17 (IX)

Să se afişeze pe ecran toate caracterele unui fişier text existent (de exemplu propriul program), caracterele netipăribile (cu cod ASCII mai mic decât 32) fiind înlocuite la scriere cu codul lor ASCII, precedat de caracterul '#'.

7.18 (IX)

Să se creeze un fişier text având ca sursă două fişiere text (eventual programe existente în directorul curent), al doilea fiind adăugat în continuarea primului omiţând doar liniile din al doilea fişier care coincid cu cel puţin o linie din primul fişier.

7.19 (IX)

Utilizaţi unitul DOS pentru a determina lungimea totală a fişierelor din directorul curent.

7.20 (IX)

Fişierul 'unitati.dat' conţine numele şi capitalul în lei ale fiecărei unităţi economice dintr-o ramură. Pentru unităţile cu capital privat românesc se precizează numele patronului, pentru cele cu capital mixt (româno-...) se precizează ţara parteneră şi procentul deţinut de aceasta, iar pentru cele cu capital de stat se precizează numărul de angajaţi.

a) Să se elimine din fişier unităţile care dispun de un capital sub 1000000;
b) Să se afişeze conţinutul fişierului pe cele trei categorii de unităţi;
c) Să se adauge *n* înregistrări noi la sfârşitul fişierului;
d) Să se ordoneze înregistrările în ordinea descrescătoare a capitalurilor;
e) Să se modifice o înregistrare dată prin numărul ei de ordine (1, 2, ...).

7.21 (IX)

Să se realizeze o copie a unui fişier oarecare din directorul curent în directorul rădăcină al discului curent, controlând ca fişierele să aibă exact aceeaşi lungime. Să se simuleze între timp pe ecran procesul de copiere printr-o reprezentare grafică sugestivă.

Observaţie: copierea cu blocuri de câte un octet este dezastruoasă pentru fişierele mari; de aceea se recomandă copierea folosind blocuri mari, iar pentru octeţii rămaşi necopiaţi se închide şi se redeschide fişierul folosind blocuri mai mici:

```
reset(f,1);l:=filesize(f);
reset(f,2000);rewrite(g,2000);
for i:=1 to l div 2000 do
    begin blockread(f,buff,1,n);...end;
reset(f,1);reset(g,1);
seek(f,l div 2000*2000);seek(g,filesize(g));
for i:=1 to l mod 2000) do
    begin blockread(f,x,1,n).....end;
```

Sau – ca o altă modalitate – se foloseşte semnificaţia parametrului opţional *n* din procedura *blockread* pentru a calcula numărul de octeţi rămaşi necopiaţi (a se vedea [3], vol. 2).

Scanned with OKEN Scanner

VIII. RECURSIVITATE

Probleme rezolvate

8.1 (V)
Să se calculeze cel mai mare divizor comun a două numere naturale.

R 8.1
Vom folosi Algoritmul lui Euclid, în varianta sa recursivă:

$$cmmdc(a,b)=\begin{cases} nedefinit \text{ dacă } a=0 \text{ şi } b=0 \\ b \text{ dacă } a=0 \\ a \text{ dacă } b=0 \\ cmmdc(b, a \bmod b) \end{cases}$$

```
program divizor;
var x,y,c:word;
function cmmdc(a,b:word):word;
begin
  if (a=0)and(b=0) then cmmdc:=0
  else if b=0 then cmmdc:=a
       else if a=0 then cmmdc:=b
            else cmmdc:=cmmdc(b,a mod b);
end;
begin
  write('Introuceti numerele : ');
  readln(x,y);
  c:=cmmdc(x,y);
  if c=0 then writeln('Nu este definit cmmdc(0,0) !')
  else writeln('cmmdc(',x,',',y,')=',c)
end.
```

Observaţie: Această definiţie recursivă a Algoritmului Euclid nu este unică. Pentru $a, b > 0$ de exemplu, se poate defini:

```
cmmdc(a,b)=if a=b then a
           else if a>b then cmmdc(a-b,b)
                else cmmdc(a,b-a)
```

8.2 (X)
Să se genereze permutările mulţimii $\{1, 2, ..., n\}$, cu n fixat.

R 8.2
Vom folosi o metodă recursivă de construcţie a permutărilor. Permutarea este generată în vectorul a, de către procedura *rec*, al cărei unic parametru x reprezintă poziţia curentă, care trebuie completată. Vectorul b se foloseşte pentru a marca elementele deja utilizate până la pasul curent.

Scanned with OKEN Scanner

```
program perm;
var a,b:array[1..50] of byte;
    i,n:byte;
procedure rec(x:byte);
var i:byte;
begin
  if x>n then
     begin
          for i:=1 to n do write(a[i],' ');
          writeln;
          exit
  end;
  for i:=1 to n do
      if b[i]=0 then begin
         b[i]:=1;
         a[x]:=i;
         rec(x+1);
         b[i]:=0
      end
end;
begin
  write('Introduceti dimensiunea multimii : ');
  readln(n);
  for i:=1 to n do b[i]:=0;
  writeln('Permutarile sunt :');
  rec(1)
end.
```

8.3. (VIII)

Se dă o fotografie specificată printr-o matrice pătratică care conţine 0 şi 1 (0 pentru punctele albe, 1 pentru punctele negre). Se consideră fondul alb, obiectele negre, iar dacă două puncte negre sunt vecine pe linie, coloană sau diagonală atunci aparţin aceluiaşi obiect. Să se numere câte obiecte distincte apar în fotografie.

R 8.3

Algoritmul folosit este următorul: se caută un punct negru nemarcat, se marchează cu 2, apoi marcajul se repetă recursiv pentru fiecare dintre vecinii săi negri. În urma acestui procedeu obiectul din care face parte punctul, este complet marcat cu 2. În continuare este căutat următorul punct negru, şi metoda de marcare este repetată (Atenţie! punctele marcate cu 2 nu sunt considerate în continuare ca fiind negre). De fiecare dată când este găsit un punct negru nemarcat este incrementat contorul de obiecte (o). Citirea datelor se face dintr-un fişier pe a cărui primă linie se află dimensiunea matricii – imagine, iar pe liniile următoare matricea propriu-zisă.

Observaţii:

1. Poziţia relativă a tuturor vecinilor unui punct este memorată în vectorii vx şi vy.

2. Deşi elegantă, soluţia este mare consumatoare de memorie pentru stivă (pentru un obiect cu p puncte sunt necesari $p*3$ octeţi pentru stivă; de aceea a fost necesară specificarea explicită a dimensiunii stivei).

Scanned with OKEN Scanner

```
{$m 60000,0,4000}
program fotografie;
const nmax=100;
      vx:array[1..8] of integer=(-1, 0, 1,1,1,0,-1,-1);
      vy:array[1..8] of integer=(-1,-1,-1,0,1,1, 1, 0);
var n,i,j,o:byte;
    f:text;
    a:array[1..nmax,1..nmax] of 0..2;
procedure marcheaza(x,y:byte);
var i:byte;
begin
  if (x>0)and(x<=n)and(y>0)and(y<=n) then
    if a[x,y]=1 then
       begin
            a[x,y]:=2;
            for i:=1 to 8 do
                marcheaza(x+vx[i],y+vy[i])
       end
end;
begin
  assign(f,'date.in');
  reset(f);
  readln(f,n);
  for i:=1 to n do
      for j:=1 to n do
          read(f,a[i,j]);
  close(f);
  o:=0;
  for i:=1 to n do
      for j:=1 to n do
          if a[i,j]=1 then
             begin
                  inc(o);
                  marcheaza(i,j)
             end;
  writeln('Numarul de obiecte din fotografie : ',o)
end.
```

8.4. (VIII)

Să se plaseze pe o tablă de şah 8 dame astfel încât să nu se atace reciproc. Să se genereze toate soluţiile.

R 8.4

Soluţia folosită se bazează pe metoda **backtraking** şi este realizată recursiv. Se poate observa cu uşurinţă că pe fiecare linie trebuie să stea o singură damă. Procedura *pune_damă* are ca parametru linia pe care trebuie să se aşeze o damă astfel încât să nu fie atacată de damele poziţionate anterior. Pentru fiecare poziţie corectă găsită, procedura se autoapelează pentru linia următoare. De remarcat este modul în care se memorează tabla (vectorul *a* cu semnificaţia că *a[i]* este coloana pe care se plasează dama de pe linia *i*) şi simplitatea condiţiei ca două dame să nu se atace.

Scanned with OKEN Scanner

```
program dame;
const nmax=20;
var n,i,j:byte;
    a:array[1..nmax] of 0..nmax;
    nrs:word;    {numar solutii}
procedure afis;
var i,j:byte;
begin
  for i:=1 to n do begin
    for j:=1 to n do
      if a[i]=j then write('* ')
      else write('o ');
    writeln
  end;
  writeln;
  nrs:=nrs+1
end;
procedure pune_dama(x:byte);
var i,j:byte;v:boolean;
begin
  if x>n then afis
  else for i:=1 to n do begin
    v:=true;
    for j:=1 to x-1 do
        if (a[j]=i) or (abs(j-x)=abs(a[j]-i))
          then v:=false;
        if v then begin
           a[x]:=i;
           pune_dama(x+1);
        end;
  end;
end;
begin
  Write('Introduceti dimensiunea tablei : ');
  readln(n);
  nrs:=0;
  pune_dama(1);
  Writeln('Numarul de solutii : ',nrs)
end.
```

8.5. (IX)

Să se genereze toate partiţiile mulţimii $\{1, 2, ..., n\}$ cu n fixat.

R 8.5

Vom folosi o procedură recursivă cu un singur parametru (elementul curent) care trebuie distribuit în una dintre submulţimi. Strategia de distribuire este următoarea: elementul curent este distribuit pe rând în fiecare dintre submulţimile deja formate şi în cele din urmă este creată o nouă submulţime care îl are doar pe el ca element. Semnificaţia variabilelor este următoarea: *nrp* – numărul curent de submulţimi, *a* – vector în care *a*[*i*] este numărul submulţimii căreia îi aparţine elementul *i*.

Scanned with OKEN Scanner

```
program partitie;
var i,j,n,nrp:byte;
    a:array[1..100] of byte;

procedure afis;
var i,j:byte;
begin
  write('(');
  for i:=1 to nrp do begin
    write('(');
    for j:=1 to n do
        if a[j]=i then write(j,',');
    write(#8,'),')
  end;
  writeln(#8,')')
end;

procedure gen(x:byte);
var i:byte;
begin
  if x>n then afis
  else begin
    for i:=1 to nrp do begin
      a[x]:=i; gen(x+1)
    end;
    nrp:=nrp+1; a[x]:=nrp;
    gen(x+1);
    nrp:=nrp-1
  end
end;

begin
  write('Introduceti nr.de elmente : '); readln(n);
  nrp:=0; gen(1)
end.
```

8.6 (VIII)

Se dau n tipuri de monezi. Să se plătească o sumă dată s, folosind un număr minim de monezi din tipurile date. Se consideră că există un număr suficient de monezi din fiecare tip.

R 8.6

Soluţia este **backtracking**. Procedura *rec* primeşte ca parametru un tip de monedă şi încearcă să-l folosească pentru plata sumei rămase până la momentul curent, pornind de la numărul maxim de monezi pe care le poate folosi şi până la 0, procedura apelându-se recursiv pentru moneda următoare. De remarcat că în *s* este memorată suma care mai este de plătit la momentul curent, în *nr* numărul de monezi deja folosite, în *b* soluţia curentă, în *nrm* numărul minim de monezi dintre soluţiile obţinute, şi în *bm* soluţia cea mai bună obţinută până la momentul curent. Se poate observa o îmbunătăţire a algorimului clasic: dacă în soluţia curentă s-au folosit deja mai multe monezi decât în soluţia cea mai bună de până în prezent, căutarea este abandonată, revenindu-se la pasul anterior (condiţia de la începutul procedurii *rec*).

Scanned with OKEN Scanner

```
program suma;
const nmax=100;
var i,n,s:word;
    a,b,bm:array[1..nmax] of word;
    nr,nrm:word;
procedure rec(x:word);
var i:integer;
begin
  if (nr<=nrm)or(nrm=0) then begin
    if x>n then begin
      if (s=0)and((nr<nrm)or(nrm=0)) then begin
        bm:=b;nrm:=nr;
      end;
    end else for i:=s div a[x] downto 0 do begin
        s:=s-a[x]*i;b[x]:=i;nr:=nr+i;
        rec(x+1);
        s:=s+a[x]*i;nr:=nr-i;
    end
  end;
end;
begin
  write('Introduceti numarul de tipuri de monezi : ');readln(n);
  writeln('Introduceti valorile monezilor : ');
  for i:=1 to n do read(a[i]);
  write('Introduceti suma de platit : ');readln(s);
  nr:=0;nrm:=0;
  rec(1);
  if nrm=0 then begin
    if s=0 then writeln('Nu este necesara nici o moneda !')
           else writeln('Nu se poate plati suma data !');
  end else begin
    writeln('Numarul minim de monezi este : ',nrm);
    for i:=1 to n do if bm[i]<>0 then
      writeln(bm[i],' monezi de ',a[i],' lei')
  end
end.
```

8.7 (VIII)

Se consideră trei tije, pe prima dintre ele fiind aşezate n discuri de dimensiuni diferite, astfel încât cel mai mare este jos, şi, peste el, în ordine descrescătoare a dimensiunii, celelalte discuri. Discurile pot fi mutate de pe o tijă pe alta, luând întotdeauna un disc care nu are deasupra sa alte discuri şi având grijă ca niciodată să nu fie aşezat un disc mai mare peste unul mai mic. Să se scrie un program care generează o secvenţă de mutări prin care discurile sunt mutate de pe tija 1 pe tija 2.

R 8.7

Vom folosi o strategie de tip **divide et impera**. Astfel, pentru a muta n discuri de pe prima tijă pe a doua, trebuie să mutăm primele $n - 1$ discuri de pe prima tijă pe a treia, apoi să mutăm discul n de pe prima tijă pe a doua, şi în cele din urmă să mutăm cele $n - 1$ discuri de pe a treia tijă pe a doua. Pentru a muta cele $n - 1$ discuri, se mută întâi primele $n - 2$ etc. Strategia este ilustrată în

Scanned with OKEN Scanner

programul următor. Procedura *hanoi* primeşte ca parametru numărul de discuri pe care trebuie să le mute, tija de plecare şi tija destinaţie. De remarcat că dacă i şi j sunt numerele a două tije, deoarece suma numerelor de ordine ale tuturor tijelor este $1 + 2 + 3 = 6$, atunci $6 - i - j$ este numărul celei de-a treia tije.

```
program turnuri_hanoi;
var n:byte;
procedure hanoi(n,i,j:byte);
begin
  if n>0, then
     begin
          hanoi(n-1,i,6-i-j);
          writeln('Se muta discul ',n,' de pe tija ',i,' pe ',j);
          hanoi(n-1,6-i-j,j)
    end
end;
begin
  Write('Introduceti numarul de discuri : ');
  readln(n);
  hanoi(n,1,2)
end.
```

8.8 (**X – C**, Concurs ACM, 1993)

Se dă un număr natural n ($4 \le n \le 14$). Să se genereze cel mai scurt şir format din 0 şi 1, de lungime $2^n + n - 1$, astfel încât oricare ar fi două secvenţe de lungime n ale acestui şir, acestea să fie diferite.

R 8.8

Soluţia este de tip **backtracking**. În vectorul e este generat şirul iar în vectorul v se păstrează valorile secvenţelor de biţi consecutivi, sub formă de numere zecimale.

```
program subsiruri_biti;
uses crt;
var v:array[0..16383]of word;
    e:array[0..16383]of boolean;
    n:byte; m,cr:word;
 procedure construieste;
{Construirea succesiunii de val. zecimale prin metoda backtracking}
  var x,i:word;
  begin
    for i:=0 to m do
        begin
             e[i]:=false;
             v[i]:=0
        end
    v[0]:=0;e[0]:=true;
    for i:=1 to n do
        begin
             e[1 shl (i-1)]:=true;
             v[i]:=1 shl (i-1)
        end;
    cr:=n+1;
```


Scanned with OKEN Scanner

```
  while cr<=m do begin
      x:=v[cr-1]*2 mod (m+1);
      if e[x] then
        if e[x+1] then begin
           repeat
                dec(cr); e[v[cr]]:=false;
           until (v[cr] mod 2=0)and not e[v[cr]+1];
           v[cr]:=v[cr]+1; e[v[cr]]:=true
        end
        else begin
             v[cr]:=x+1;e[x+1]:=true
        end
      else begin
        v[cr]:=x; e[x]:=true
      end;
      inc(cr)
   end
 end;
procedure scrie;                          {Scrie rezulatul}
 var i,j:word;
 begin
   writeln('Cel mai mic subsir de ',m+n,' biti este:');
   for i:=1 to n-1 do write('0');
   for i:=0 to m do begin
       if (i+n-1) mod 60=0 then writeln;
       write(v[i] mod 2)
   end;
   writeln;
   writeln('Valorile corespunzatoare ale subsirurilor sunt:');
   for i:=0 to m div 8-1 do begin
       for j:=0 to 7 do write(v[8*i+j]:6);
       writeln
   end;
   for i:=m div 8*8 to m do write(v[i]:6);
   writeln
 end;
begin   {Program principal}
  write('Introduceti n : ');
  readln(n);
  m:=1 shl n -1;
  construieste;
  scrie;
  writeln('Gata');
  readln
end.
```

8.9 (**X** – C, Concurs „Grigore Moisil", Lugoj, 1995)

Din fişierul de intrare, al cărui nume se dă la tastatură, se citesc perechi de coordonate reprezentând vârfurile opuse ale unor dreptunghiuri, de forma *x1 y1 x2 y2* ($1 \le x \le 25$, $1 \le y \le 80$). Citirea se face până la întâlnirea sfârşitului de fişier. Prin suprapunerea acestor dreptunghiuri, pe ecran se vor obţine figuri conexe formate din dreptunghiuri suprapuse. Să se deseneze pe ecranul calculatorului numai contururile exterioare ale acestor figuri.

Scanned with OKEN Scanner

R 8.9

Ecranul (alfanumeric) va fi memorat în tabloul *a*, de tip caracter, care va fi iniţial umplut cu spaţii. În acest tablou se marchează ramele dreptunghiurilor cu caracterul "#". În continuare se umple spaţiul exterior dreptunghiurilor cu caracterul „e", şi, în cele din urmă vor fi afişate doar acele caracter din ramele dreptunghiurilor care au un vecin de tip „e", adică exact contururile cerute. Pentru implementarea mai uşoară a marcării exteriorului, tabloul a a fost bordat cu o margine formată din spaţii. Procedura *umple* primeşte ca parametri coordonatele unui punct pe care îl marchează cu „e", şi se apelează pentru toţi vecinii săi de tip spaţiu.

```
{$m 64000,0,64000}

program dreptunghiuri;
const           {coordonatele relative ale vecinilor unui punct}
  vx:array[1..8] of integer=(-1,-1,-1,0,1,1,1,0);
  vy:array[1..8] of integer=(-1,0,1,1,1,0,-1,-1);
var s:string;f:text;
    x1,y1,x2,y2:integer;
    a:array[0..25,0..81] of char;
    i,j,k,t:integer;
    margine:boolean;

procedure umple(x,y:integer);
var i:integer;
begin
  if (x>=0)and(x<=25)and(y>=0)and(y<=81) then
    {se testeaza ca punctul sa fie in ecran}
    if a[x,y]=' ' then begin {se testeaza ca punctul sa fie liber}
      a[x,y]:='e';           {se marcheaza cu "e"}
      for i:=1 to 8 do umple(x+vx[i],y+vy[i])
    {se apeleaza pentru vecinii punctului (x,y)}
    end
end;

begin
  Write('Introduceti numele fisierului de intrare : ');
  readln(s);
  assign(f,s);
  reset(f);
  for i:=0 to 25 do for j:=0 to 81 do a[i,j]:=' ';
  while not seekeof(f) do begin
    readln(f,x1,y1,x2,y2);
    if x1>x2 then begin t:=x1;x1:=x2;x2:=t end;
         {Se inverseaza coordonatele}
    if y1>y2 then begin t:=y1;y1:=y2;y2:=t end;
         {a.i. x1 si y1 sa fie cele mai mici}
    for i:=x1 to x2 do begin
      a[i,y1]:='#';a[i,y2]:='#'      {Se marcheaza    }
    end;
    for j:=y1 to y2 do begin          {marginile dreptunghiurilor}
      a[x1,j]:='#';a[x2,j]:='#'
    end
  end;
```


Scanned with OKEN Scanner

```
  umple(0,0); {Se umple zona exterioara dreptunghiurilor cu "e".}
  {In continuare se vor afisa numai ramele de dreptunghiuri care
  au "contact" cu exteriorul (marcat acum cu "e")                }
  writeln;
  for i:=1 to 24 do for j:=1 to 80 do begin
    margine:=false;
    {margine = true daca (i,j) are un vecin de tip "e" (exterior) }
    for k:=1 to 8 do if a[i+vx[k],j+vy[k]]='e' then margine:=true;
       {s-au testat vecinii lui (i,j)}
    if (a[i,j]='#')and(margine) then write('#') else write(' ')
  end
end.
```

8.10 (**X – C**, Judeţul Alba, Etapa judeţeană, 1994)

Se consideră o stivă de case de bani sub formă de piramidă. Astfel pe primul nivel vor fi n case, pe următorul $n-1$ case, apoi $n-2$, .., până la ultimul nivel unde va fi o sigură casă de bani. Deci fiecare casă se va „sprijini" pe două case de bani din etajul anterior, mai puţin cele din etajul de bază care se vor sprijini direct pe podea, în total fiind $\frac{n(n+1)}{2}$ case de bani.

Se dau $\frac{n(n+1)}{2}$ saci cu bani, fiecare conţinând o anumită sumă. Să se distribuie aceşti saci, fiecare într-o casă, astfel încât în fiecare casă să fie exact atâţia bani cât în cele două cutii pe care se sprijină la un loc. Pentru casele din stratul de bază nu există nici o condiţie.

Datele vor fi citite dintr-un fişier al cărui nume este introdus de la tastatură, cu următorul format:

```
n
s1 s2 s3 ... s(n+1)*n/2
```

unde n este numărul de cutii din stratul de bază, iar s_1, s_2... sunt sumele de bani din fiecare sac. Rezultatul va fi afişat pe ecran sub forma unei piramide în care sunt trecute sumele de bani din fiecare cutie.

R 8.10

Rezolvarea este de tip **backtracking**. Vom porni de la o piramidă iniţial goală, în care procedura *pune* va introduce săculeţii. Sortând sacii înainte de apelul lui *pune*, în ordine descrescătoare a valorii conţinută în ei, vom avea următoarele restricţii în poziţionarea sacilor:

– Nu se poate poziţiona un sac dacă în una dintre cutiile de deasupra lui se găseşte o cantitate de bani mai mică sau egală cu cea din sacul curent. Această condiţie rezultă chiar din enunţul problemei.

– Nu se poate introduce un sac dacă în una din cutiile de deasupra lui nu se găseşte nimic. Aceasta deoarece în continuare nu sunt decât saci cu o cantitate cel mult egală cu cea din sacul curent, deci nu vom avea ce să punem în cutia de deasupra.

– Dacă în una dintre cutiile alăturate sacului curent se află deja introdus un sac, atunci vom verifica să fie respectată condiţia ca suma celor doi saci să se regăsească în cutia de deasupra lor.

Scanned with OKEN Scanner

Având în vedere aceste condiţii, numărul de posibilităţi de alegere la fiecare pas se reduce drastic, faţă de varianta **backtracking** clasică (implicit şi timpul de execuţie al programului).

Observaţie: Referitor la implementare, se poate remarca folosirea variabilei globale *ver*, în procedura recursivă *pune*. Acest lucru are ca avantaj economisirea stivei, critică în cazul de faţă, datorită numărului destul de mare de apeluri imbricate ale procedurii *pune*. În general aceste metode nu sunt recomandate, datorită scăderii clarităţii programului şi a faptului că sunt generatoare de erori logice, foarte greu de detectat. Astfel, după secvenţa:

```
if ver then begin
   a[i,j]:=b[s];
   pune(s+1);a[i,j]:=0
end;
```

variabila *ver* va avea o valoare greu previzibilă, iar dacă mai este necesară valoarea ei, aceasta trebuie recalculată (în programul nostru valoarea ei nu mai este necesară după testul din IF).

```
program saculeti;
const nmax=100;
var s:string; f:text; ver:boolean;
    i,j,t,n,m:integer;
    b:array[1..nmax*(nmax+1) div 2] of integer;
    a:array[1..nmax, 1..nmax] of integer;
procedure afis;
var i,j:integer;
begin
  for i:=1 to n do begin
    for j:=1 to n-i div 2 do write('      ');
    if odd(n-i) then write('   ');
    for j:=1 to i do write(a[i,j]:6);
    writeln;
  end;
end;
procedure pune(s:integer);
var i,j:byte;
begin
  if s>m then begin afis;halt; end
  else begin
    for i:=1 to n do for j:=1 to i do
      if a[i,j]=0 then begin
        ver:=true;
        if i>1 then begin
          if j<i then
            if (a[i,j+1]=0)and(a[i-1,j]<=b[s])or
               (a[i,j+1]<>0)and(a[i-1,j]<>b[s]+a[i,j+1])
            then ver:=false;
          if j>1 then
            if (a[i,j-1]=0)and(a[i-1,j-1]<=b[s])or(a[i,j-1]<>0)
               and(a[i-1,j-1]<>b[s]+a[i,j-1]) then ver:=false;
        end;
        if ver then begin
          a[i,j]:=b[s];
          pune(s+1); a[i,j]:=0
        end
      end
  end
end;
```


Scanned with OKEN Scanner

```
begin
  write('Introduceti numele fisierului : ');readln(s);
  assign(f,s);reset(f);
  read(f,n);m:=n*(n+1) div 2; {m - numarul de saculeti}
  for i:=1 to n*(n +1) div 2 do read(f,b[i]);
  close(f);
  for i:=1 to m-1 do     {sortarea sacilor in ordine descrescatoare}
    for j:=i+1 to m do   {a cantitatii de bani continuta            }
      if b[i]<b[j] then begin
        t:=b[i];b[i]:=b[j];b[j]:=t;
      end;
  pune(1);
  writeln('Nu exista solutie !')
end.
```

8.11 (VIII)

Se dau două şiruri de caractere. Să se determine dacă ele sunt anagrame, adică sunt formate din aceleaşi litere dar puse în altă ordine.

R 8.11

Vom da o soluţie recursivă care funcţionează astfel: se ia primul caracter din primul şir şi se caută în cel de-al doilea şir. Dacă nu este găsit, cele două şiruri nu sunt anagrame; dacă este găsit, este eliminat din amândouă şi procedura se apelează recursiv pentru şirurile astfel obţinute.

Dacă cele două şiruri devin vide simultan, atunci ele sunt anagrame; în caz contrar răspunsul este negativ.

```
program anagramari;
type sir=string[30];
var s,t:sir;
function anagram(s,t:string):boolean;
var i:byte;
begin
  if s='' then if t='' then anagram:=true
                       else anagram:=false
  else begin
    i:=pos(s[1],t);
    if i=0 then anagram:=false
    else begin
      delete(s,1,1);
      delete(t,i,1);
      anagram:=anagram( s,t)
    end
  end
end;
begin
  write('Introduceti primul sir de caractere     : ');readln(s);
  write('Introduceti al doilea sir de caractere  : ');readln(t);
  if anagram(s,t) then  writeln('Cele doua siruri sunt anagrame !')
  else writeln('Cele doua siruri nu sunt anagrame !')
end.
```

O variantă de implementare a aceleiaşi idei constă în folosirea lui *s* şi *t* drept variabile globale. În acest caz memoria de tip stivă ocupată este mai mică iar viteza de execuţie mai mare.

Scanned with OKEN Scanner

```
program anagramari;
type sir=string[30];
var s,t:sir;
function anagram:boolean;
var i:byte;
begin
  if s='' then if t='' then anagram:=true
                       else anagram:=false
  else begin
    i:=pos(s[1],t);
    if i=0 then anagram:=false
    else begin
      delete(s,1,1);
      delete(t,i,1);
      anagram:=anagram
    end
  end
end;
begin
  write('Introduceti primul sir de caractere     : ');
  readln(s);
  write('Introduceti al doilea sir de caractere  : ');
  readln(t);
  if anagram then  writeln('Cele doua siruri sunt anagrame !')
  else writeln('Cele doua siruri nu sunt anagrame !')
end.
```

8.12 (**VIII** – C, Concurs PACO, Bucureşti, 1995)

Se dă o listă formată din n cuvinte. Să se formeze cu ele cel mai lung şir în care fiecare cuvânt începe cu litera cu care se termină predecesorul său (mai puţin primul cuvânt care poate să fie oricare). Datele se citesc dintr-un fişier al cărui nume se introduce de la tastatură, cu următorul format:

```
n
c1
c2
.
.
cn
```

unde n – numărul de cuvinte, şi c_1 ... c_n – cuvintele.

R 8.12

Soluţia este de tip **backtracking** clasic. Noutatea constă numai în organizarea datelor. S-a optat pentru o structură mai complicată, dar mult mai eficientă din punctul de vedere al timpului de acces. Astfel, în loc să se memoreze o simplă listă de cuvinte, în care toate căutările ar fi secvenţiale, vom memora într-un tablou separat ultima literă a fiecărui cuvânt (b), iar în alt tablou cuvintele care încep cu o anumită literă (perechea c, d – în $c[i]$ numărul de cuvinte care încep cu litera de cod i, iar în $d[i]$ – lista acestor cuvinte). În acest mod, căutarea cuvintelor care pot succede un cuvânt dat va fi mult mai rapidă (nu va mai presupune parcurgerea întregului tablou de cuvinte).

Scanned with OKEN Scanner

```
program cuvinte;
const nmax=30;
var
   a:array[1..nmax] of string; {cuvintele}
   b:array[1..nmax] of byte; {ultima litera a fiecaruia}
   c:array[0..255] of byte; {cate cuvinte incep cu o anumita litera}
   d:array[0..255,1..nmax] of byte; {cuvintele care incep ...}
   e:array[1..nmax] of boolean; {Au fost utilizat sau nu}
   sol,solm:array[0..nmax] of byte; {Solutia curenta, optima}
   nrm:byte; { numarul de elemente din sol curenta, optima }
   f:text;s:string;
   i,j,n:byte;

procedure rec(x:integer);
var i:integer;sfarsit:boolean;ulti:byte;
{In ulti se va introduce ultima litera a cuvantului anterior}
begin
  if x=1 then
    for i:=1 to n do begin
      e[i]:=true; sol[x]:=i;
      rec(x+1);
      e[i]:=false
    end
  else begin
    sfarsit:=true; ultimalitera:=b[sol[x-1]];
    for i:=1 to c[ulti] do
      if not e[d[ulti,i]] then begin
        sfarsit:=false; e[d[ulti,i]]:=true;
        sol[x]:=d[ulti,i];
        rec(x+1);
        e[d[ulti,i]]:=false
      end;
    if sfarsit then if x-1>nrm then begin
      solm:=sol;nrm:=x-1
    end
  end
end;

begin
  write('Introduceti numele fisierului : ');readln(s);
  assign(f,s);reset(f);
  readln(f,n);
  for i:=1 to n do readln(f,a[i]);
  for i:=0 to 255 do c[i]:=0;
  for i:=1 to n do begin
    b[i]:=ord(a[i][length(a[i])]);
    inc(c[ord(a[i,1])]);
    d[ord(a[i,1]),c[ord(a[i,1])]]:=i;
    e[i]:=false;
  end;
  nrm:=0;
  rec(1);
  writeln('Solutia cea mai buna este : ');
  for i:=1 to nrm do write(a[solm[i]],' ');
  writeln
end.
```


Scanned with OKEN Scanner

Probleme propuse

8.13 (VIII)

Să se construiască o procedură recursivă pentru determinarea elementului minim dintr-un tablou cu maxim 100 elemente numere reale.

8.14 (X)

Se dă următoarea funcţie definită recursiv:

```
f(x,y)=if x=0 then y+1
          else if y=0 then f(x-1,1)
                      else f(x-1,f(x,y-1))
```

unde x, y iau valori numere întregi (funcţia Ackermann).

Să se realizeze o procedură recursivă care să calculeze această funcţie şi să se determine f(1, n), f(2, n) pentru n citit de la tastatură.

8.15 (X)

Să se construiască o procedură recursivă pentru calculul combinărilor C_n^k ($0 \le k \le n \le 10000$) cu n, k date la tastatură.

8.16 (X)

Să se scrie o procedură recursivă care şterge un fişier dat din directorul curent şi din toate subdirectoarele directe şi indirecte ale acestuia.

Indicaţie: Se va folosi unitul DOS ştiind că atributul unui director este 12.

Scanned with OKEN Scanner

IX. GRAFURI

Probleme rezolvate

Comentariu: Pentru complementele teoretice necesare înţelegerii teoriei grafurilor, se poate folosi proiectul de manual Cornelia Ivaşc, Mona Prună – Bazele Informaticii pentru clasa X, Editura Petrion, 1995

9.1 (X)

Se consideră un graf şi un nod – considerat ca nodul numărul 1; se cere să se determine toate nodurile accesibile din acest nod.

R 9.1

Structura grafului se precizează prin numărul de noduri şi legăturile (arcele) existente între oricare două noduri astfel: în matricea de adiacenţă a, $a[i, j] = 0$ dacă nu există arc de la i la j şi $a[i, j] = 1$ dacă există arc de la nodul i la nodul j.

```
program acces;
var a:array[1..20,1..20] of integer;
    acc:array[1..20] of integer;
    n,i,j,k,p:integer;
    ff:boolean;
begin
     write('dati numarul de noduri; n=');
     readln(n);
     writeln('dati matricea de adiacenta:');
     for i:=1 to n do
         for j:=1 to n do read(a[i,j]);
     acc[1]:=1;
     k:=1;
     ff:=true;
     i:=1;
     while i<=k do
           begin
                for j:=1 to n do
                    if a[i,j]=1 then
                       begin
                            for p:=1 to k do
                            if acc[p]=j then ff:=false;
                            if ff then
                               begin
                                    k:=k+1;
                                    acc[k]:=j
                               end
                       end;
                       i:=i+1
           end;
     if k=n then writeln('Toate nodurile sunt accesibile.')
            else for i:=1 to k do write(acc[i],',');
end.
```

Observaţie: Vă recomandăm Pascal, să încercaţi şi o versiune recursivă a acestei rezolvări.

Scanned with OKEN Scanner

9.2 (X)

Pentru un grup de persoane – pe care le considerăm numerotate cu numere întregi de la 1 la n se precizează relaţiile de simpatie în felul următor: pentru fiecare persoană i se citesc numerele de ordine ale persoanelor pe care aceasta le simpatizează, numere care se dau pe aceeaşi linie despărţite prin spaţii. Să se precizeze dacă în grupul de persoane amintit, toate simpatiile sunt reciproce.

Exemplu: pentru $n = 4$, un set de date de intrare poate fi

```
1: 2 4.                    {1 simpatizeaza pe 2 si 4}
2: 1                       {2 simpatizeaza pe 1}.
3: 2 4                     {3 simpatizeaza pe 2 si 4}
4: 1 3                     {4 simpatizeaza pe 1 si 3}
```

Observaţie: După cum se vede din exemplu, simpatiile nu sunt reciproce deoarece 2 nu-l simpatizează pe 3, deşi 3 îl simpatizează pe 2.

R 9.2

Se citesc în această manieră datele referitoare la un graf sub forma unor liste de adiacenţă. Propunem ca o metodă programatică pentru teoria grafurilor construirea matricei de adiacenţă şi rezolvarea cerinţei prin testarea proprietăţii de graf neorientat (în care matricea de adiacenţă este simetrică faţă de diagonala principală): $a_{ij} = a_{ji}$, oricare $1 \le i, j \le n$.

```
program liste_ad;
const max=30;
type mat_adiacenta=array[1..max,1..max]of byte;
var a:mat_adiacenta;
    n,i,j:1..max;
    este:boolean;
begin
     write('Numar de persoane:');
     readln(n);
     fillchar(a,sizeof(a),0);
     for i:=1 to n do
         begin
              write(i,': ');
              while not seekeoln do
                    begin
                         read(j);
                         a[i,j]:=1
                    end;
              readln
         end;
     este:=true;
     for i:=1 to n-1 do
         for j:=i+1 to n do
             if a[i,j]<>a[j,i] then
                begin
                     este:=false;
                     writeln('Simpatie nereciproca: ',i,'--',j)
                end;
     if este then writeln('Toate simpatiile sunt reciproce')
end.
```


Scanned with OKEN Scanner

9.3 (X)

Speologii au cercetat n culoare subterane pentru a stabili dacă aparţin aceleiaşi peşteri. Prin tehnici specifice de curenţi de aer şi de colorare a cursurilor de apă, a fost demonstrată existenţa unor canale de legătură între mai multe culoare. Precizându-se perechile de culoare între care au fost stabilite legături, să se afle dacă sistemul de culoare corespunde unei singure peşteri.

R 9.3

Perechile de culoare legate prin canale de aerisire sunt muchii ale unui graf a cărui proprietate de conexitate este răspunsul pentru problema noastră. Construim matricea de adiacenţă şi marcăm prin procedeul *Lee* toate nodurile adiacente sau tranzitiv adiacente cu un vârf oarecare (de exemplu vârful 1) verificând în final dacă mai există vârfuri nemarcate. Vectorul b memorează marcajele (0 = nemarcat, 1 = marcat):

```
program pestera;
var n,m,i,j,k:integer;
    b:array[1..50]of byte;
    a:array[1..50,1..50]of byte;   {matricea de adiacenta}
procedure marcare(i:integer);
var j:integer;
begin
     b[i]:=1;
     for j:=1 to n do
         if (b[j]=0)and(a[i,j]=1) then marcare(j)
end;
begin
     fillchar(a,sizeof(a),0) {initializarea tablourilor}
     fillchar(b,sizeof(b),0)
     write('Introduceti numarul de culoare:');readln(n);
     write('Cate canale s-au descoperit?');readln(m);
     for i:=1 to m do begin
         write('Canalul ',i,' intre: ');readln(j,k);
         a[j,k]:=1;a[k,j]:=1
     end;
     marcare(1);
     for i:=1 to n do
         if b[i]=0 then begin
            writeln('Nu este o singura pestera');halt
         end;
     writeln('Este o singura pestera');
end.
```

Observaţie: testele de validare a datelor nu au fost făcute, presupunând că nu se vor furniza date eronate (culoare inexistente sau numere de culoare care solicită supradimensionarea matricei de adiacenţă).

9.4 (X)

Într-un grup de n persoane se precizează perechi de persoane care se consideră prietene. Folosind principiul că „prietenul prietenului meu mi-este prieten", să se determine grupurile cu un numar maxim de persoane între care se pot stabili relaţii de prietenie, directe sau indirecte.

Scanned with OKEN Scanner

R 9.4

În termeni de teoria grafurilor, problema se formulează astfel: fiind dat un graf neorientat, să se determine dintre componentele sale conexe pe cele cu un număr maxim de elemente.

Se implementează un algoritm asemănător algoritmului lui Kruskal pentru *arbore parțial de cost minim*. La început, se consideră toate vârfurile grafului ca fiind componente conexe distincte (pentru aceasta se foloseşte şirul C cu n elemente, unde n reprezintă numărul de vârfuri ale grafului, în care se reține pentru fiecare vârf, care este componenta conexă din care face parte); se încearcă permanent să se unească două componente conexe diferite printr-o muchie. Când nu se mai găseşte nici o muchie care are ca extremități două vârfuri din componente conexe diferite, se determină componentele conexe maximale ca fiind mulțimile de cardinal maxim formate din indici $\{i_1, i_2, ..., i_k\}$ pentru care $C[i_1]=C[i_2]=...=C[i_k]$. Am notat cu g matricea de adiacență asociată grafului.

```
program conex_max;
const nmax=100;
type graf=array [1..nmax,1..nmax] of byte;
     tablou=array [1..nmax] of integer;
var g:graf;
    i,j,k,l,n,max:integer;
    c,cc:tablou;
    modificare:boolean;
begin
     fillchar(g,sizeof(g),0);
     write('Dati numarul de persoane: ');
     readln(n);
     writeln('Dati perechile de prieteni (0 0 pentru iesire): ');
     readln(i,j);
     while (i+j<>0) do begin
           g[i,j]:=1;
           g[j,i]:=1;
           readln(i,j)
     end;
     for i:=1 to n do c[i]:=i;
     repeat                    {unirea componentelor}
           modificare:=false;
           for i:=1 to n-1 do
               for j:=i+1 to n do
                   if (c[i] <> c[j]) and (g[i,j] <> 0) then begin
                      modificare:=true;
                      l:=c[i];
                      for k:=1 to n do if c[k]=l then c[k]:=c[j];
                   end;
     until not modificare;(*)
     max:=0;                   {determinarea cardinalului maxim}
     for i:=1 to n do begin
         cc[i]:=0;
         for j:=1 to n do
             if c[j]=c[i] then inc(cc[i]);
         if cc[i]>max then max:=cc[i]
     end;
```


Scanned with OKEN Scanner

```
    for i:=1 to n do          {determinarea componentelor maximale}
        if cc[i]=max then begin
           k:=c[i];
           write('Un grup maximal este:');
           for j:=1 to n do
               if c[j]=k then begin
                  cc[j]:=0;
                  write(j,' ')
               end;
           writeln
        end
end.
```

Observaţie: pentru determinarea tuturor grupurilor de prieteni (tuturor componentelor conexe), după instrucţiunea {*} se va implementa următoarea secvenţă de program:

```
k:=1;
writeln('Componentele conexe sunt :');
for i:=1 to n do begin
    este:=false;   {var este:boolean}
    for j:=1 to n do if c[j]=i then este:=true;
    if este then begin
       write('Componenta conexa ',k,' este :  ');
       for j:=1 to n do if c[j]=i then write(j,' ');
       writeln; inc(k)
    end
end;
```

9.5 (X)

Într-o staţiune de tratament s-au întâlnit n şahişti, codificaţi prin numere naturale de la 1 la n, care au jucat p partide de şah. Pentru fiecare partidă se precizează cei doi jucători. Să se determine un grup cât mai mare de jucători care, prin jocurile pe care le-au desfăşurat, au format o grupare de tip turneu (în care fiecare a jucat cu fiecare).

R 9.5

În graful neorientat constituit din şahişti (vârfuri) şi jocurile desfăşurate (muchii), ne interesează un subgraf complet (în care între oricare două vârfuri există muchie) cu un număr maxim de vârfuri. Algoritmul construieşte gruparea prin metoda **backtracking** şi este implementat recursiv. Este deci un algoritm cu ordin de complexitate exponenţial, un algoritm mai bun rămânând ca temă de studiu pentru cititor. În vectorul c se ţine evidenţa vârfurilor selectate la un moment dat ca făcând parte dintr-o componentă completă, iar în *csol* se memorează selecţia cu un număr maxim de vârfuri.

```
program sah;
const max=100;
type graf=array [1..max,1..max] of byte;
     tablou=array [1..max] of boolean;
var g:graf;
    i,j,k,n,p,nmax:integer;
    c,csol:tablou;
    este,gata:boolean;
```


Scanned with OKEN Scanner

```
procedure compl(lung:byte);
var i,j:byte;          {construieste subgraf complet cu lung varfuri}
begin
     gata:=true;     {arata daca se mai poate extinde componenta}
     for i:=1 to n do
         if not c[i] then begin
            este:=true;
            for j:=1 to n do
                if c[j] and (g[i,j]=0) then este:=false;
            if este then begin
               gata:=false;
               c[i]:=true;compl(lung+1);c[i]:=false
            end
         end;
     if gata and (lung>nmax) then begin
        nmax:=lung;csol:=c
     end
end;
begin
     fillchar(g,sizeof(g),0);
     fillchar(c,sizeof(c),false);
     write('Dati numarul de sahisti: ');readln(n);
     write('Dati numarul de partide: ');readln(p);
     writeln('Introduceti perechile de jucatori:');
     for i:=1 to p do begin
         readln(j,k);
         g[j,k]:=1;g[k,j]:=1
     end;
     nmax:=0;
     for i:=1 to n do begin
         c[i]:=true;compl(1);c[i]:=false
     end;
     writeln('Gruparea maximala de tip turneu este formata din: ');
     for i:=1 to n do if csol[i] then write(i,' ');
     writeln
end.
```

9.6 (X)

Reţeaua de distribuire a apei calde pentru o centrală termică zonală este formată dintr-un sistem de conducte care leagă centrala de blocuri şi blocurile între ele. Blocurile din reţea sunt numerotate cu numere întregi de la 1 la n, centrala fiind punctul 0 de distribuire. Se cunosc distanţele de la centrală la blocuri precum şi distanţele între oricare două blocuri. Să se afişeze perechile de numere desemnând punctele de distribuire între care trebuie să se monteze conducte astfel încât fiecare bloc să fie alimentat cu apă caldă (nu neapărat direct de la centrală) şi lungimea totală a conductelor necesare să fie minimă.

R 9.6

Sesizăm în subtextul problemei existenţa unui graf neorientat (contrar impresiei, sensul de distribuire a apei calde nu contează din punctul de vedere al construirii conductelor). Se cere în acest caz să se determine arborele parţial de cost minim asociat acestui graf.

Scanned with OKEN Scanner

Propunem pentru rezolvare algoritmul lui **Kruskal** folosind matricea ponderilor asociată grafului: $g[i, j]$ = costul muchiei (i, j) şi se construieşte arborele parţial de cost minim (**APM**) prin matricea asociată noului graf: $apm[i, j]$ = costul muchiei (i, j) dacă există această muchie. Se foloseşte şirul c cu n elemente, în care se reţine pentru fiecare vârf i componenta conexă din care face parte nodul.

```
program conducte;
type graf=array[0..50,0..50]of real;
     tablou=array[0..50]of integer;
var apm,g:graf;  {se initializeaza cu 0}
    k,max,cost:real;
    n,i,j,si,sj,ci,cj,pas:byte;
    c:tablou;
 begin
     write('Dati numarul de blocuri:');readln(n);max:=0;
     for i:=0 to n-1 do
         for j:=i+1 to n do
             begin     {construirea matricei de adiacenta}
                  write('Punctele ',i,'-',j,' cu distanta:');
                  readln(k);if k>max then max:=k;
                  g[i,j]:=k;g[j,i]:=k
             end;
     for i:=0 to n do c[i]:=i;
     for pas:=0 to n-1 do begin
         cost:=max+1;  {costul minim al muchiilor selectabile}
         for i:=0 to n-1 do
             for j:=i+1 to n do
                 if (c[i]<>c[j])and(g[i,j]<>0)and(g[i,j]<cost) then
                    begin
                         cost:=g[i,j];
                         si:=i;sj:=j;
                    end;
         apm[si,sj]:=cost;apm[sj,si]:=cost;
         ci:=c[si];cj:=c[sj];  {unirea componentelor}
         for i:=0 to n do if c[i]=ci then c[i]:=cj;
     end;
     writeln('Se construiesc conducte intre punctele:');
     for i:=0 to n-1 do
         for j:=i+1 to n do
             if apm[i,j]<>0 then
               writeln('(',i,',',j,') de lungime',apm[i,j]);
     readln;
end.
```

Observaţie: Algoritmul lui Kruskal este aplicat pentru un graf particular în care există muchie între oricare două noduri (graf complet). Pentru a urmări cu uşurinţă soluţia problemei, reamintim algoritmul lui Kruskal:

1. Se consideră fiecare vârf ca o componentă conexă;
2. Se repetă pasul 3 de $n - 1$ ori;
3. Se alege muchia de cost minim care uneşte două noduri care fac parte din componente conexe diferite; cele două componente se unesc iar muchia respectivă se introduce în APM;
4. Dacă nu exista nici o muchie la pasul 3, înseamnă că graful iniţial nu este conex şi, conform definiţiei unui arbore, este imposibil să găsim un arbore într-un graf neconex.

Scanned with OKEN Scanner

Pasul 3 se repetă de $n - 1$ ori pentru că un arbore cu n vârfuri are $n - 1$ muchii; datorită faptului că nu avem nici un circuit (deoarece unim tot timpul componente conexe diferite) rezultă că vom obţine un arbore parţial. Luând permanent muchii de cost minim, vom obţine chiar APM.

Pentru un graf oarecare (în care pentru uniformitate nu mai folosim nodul de indice 0) se ridică problema conexităţii, dintr-un graf neconex nefiind posibilă determinarea arborelui parţial de cost minim. Precizăm diferenţele de programare a acestui caz general:

```
write('Dati numarul de varfuri: ');
readln(n);
writeln('Dati graful (o muchie si costul ei -- i j k) : ');
writeln('0 0 0 pentru iesire.');
readln(i,j,k);
while i<>0 do begin
      g[i,j]:=k;
      g[j,i]:=k
      if k>max then max:=k
      readln(i,j,k)
end;
for i:=1 to n do c[i]:=i;
for pas:=1 to n-1 do begin
    cost:=max+1;
    for i:=1 to n-1 do
        for j:=i+1 to n do
          ......
    if cost = max+1 then begin   {nu s-a gasit nici o muchie}
       writeln('Graful nu este conex-->nu se poate gasi APM!');
       halt
    end;
```

9.7 (X)

Un elev vrea să călătorească din localitatea X în localitatea Y. Dacă în ţara respectivă există n localităţi şi ştiind timpul necesar pentru a ajunge dintr-o localitate în alta (în cazul în care se poate ajunge direct) se cere să se determine timpul minim în care elevul poate să ajungă din X în Y.

R 9.7

Altfel spus, fiind dat un digraf (N – mulţimea nodurilor, A – mulţimea arcelor) şi o funcţie K : A -> $\mathbb{R}$ care asociază fiecărui arc *ponderea* sa (*costul* respectivului arc), se cere să se determine costul minim al unui drum între două vârfuri date (unde se consideră costul unui drum egal cu suma costurilor arcelor din componenţa sa).

Pentru rezolvarea acestei probleme vom folosi matricea ponderilor asociată grafului (pe care o vom citi din fişierul text *mat.dat*):

$$A[i,j] = \begin{cases} K(i,j) & \text{costul arcului } (i,j) \text{ dacă acesta există} \\ 0 \text{ dacă } i = j & \\ \infty & \text{convenţional} = -1 \text{ dacă arcul } (i,j) \text{ nu există} \end{cases}$$

Rezolvarea problemei se bazează pe algoritmul lui Dijkstra.

Scanned with OKEN Scanner

```
program dijkstra;
var a:array[1..20,1..20]of real;
    d:array[1..20]of real;
    i,j,k,n,x,y:byte;min:real;
    c:set of byte;  {multimea nodurilor neselectate}
    f:text;
begin
     assign(f,'mat.dat');reset(f);readln(f,n);
     for i:=1 to n do for j:=1 to n do read(f,a[i,j]);
     write('Localitatea 1:');
     readln(x);
     write('Localitatea 2:');
     readln(y);
     c:=[1..n]-[x];      {presupunem x<>y}
     for i:=1 to n do
         if i in c then d[i]:=a[x,i];
     for k:=1 to n-2 do begin
         min:=1e38; {min=valoare foarte mare}
         for i:=1 to n do
             if (i in c)and(d[i]>0)and(d[i]<min) then begin
                min:=d[i];j:=i
             end;
         c:=c-[j];
         if j=y then break; {instructiunea poate lipsi}
         for i:=1 to n do
             if (i in c)and(d[j]<>-1)and(a[j,i]<>-1) then
                if (d[i]=-1) or (d[i]>d[j]+a[j,i]) then
                   d[i]:=d[j]+a[j,i] {*}
     end;
     writeln(#13#10,d[y])
end.
```

Observaţie: Amintim paşii algoritmului Dijkstra:

Pasul 1 – se selectează nodul iniţial al drumului căutat calculând într-un vector d distanţele până la celelalte noduri ($d[i] = 0$ dacă şi numai dacă nu există arc către nodul i);

Pasul 2 – se repetă pasul 3 de $n-2$ ori (dacă ne interesează distanţele minime de la nodul iniţial către toate celelalte noduri) sau până ce se selectează nodul terminal;

Pasul 3 – se selectează alt nod j pentru care $d[j]$ = minim şi se recalculează distanţele la nodurile rămase $d[i] = \min(d[i], d[j] + a[j, i])$ unde j este nodul proaspăt selectat.

Construirea efectivă a unui drum de cost minim, presupune memorarea într-un tablou de n componente, pentru fiecare nod i, a predecesorului acestuia: $p[i]$ = nodul care precede pe i în drumul de cost minim. Astfel, instrucţiunii {*} i se adaugă (într-o instrucţiune compusă) `p[i]:=j`. Iniţial $p[i] = x$, oricare $i \in \{1, ..., n\}$. Scrierea drumului de lungime minimă se realizează recursiv cu procedura:

```
procedure drum(i:byte);
begin
     if i<>x then drum(p[i]);
     write(i,' ')
end;
```

apelată la sfârşitul programului principal prin:

```
drum(y);writeln
```


Scanned with OKEN Scanner

9.8 (X)

Fiind date *n* oraşe şi costurile tuturor drumurilor directe care există între oraşe, să se afle costul minim pentru a ajunge dintr-un oraş în oricare altul.

R 9.8

Modelul folosit în rezolvarea acestei probleme este un graf neorientat în care oraşele sunt vârfuri, drumurile directe sunt muchii, iar distanţele sunt ponderile muchiilor. Algoritmul propus poate fi utilizat şi pentru grafuri orientate. Pentru rezolvare vom utiliza în reprezentarea digrafului matricea ponderilor.

Observaţie: Programul reprezintă o implementare a algoritmului lui **Floyd**:

```
var a:array [1..100,1..100] of real;
    i,j,n:integer; c:real;
procedure calculeaza_matricea_drumurilor_minime;
 var k,i,j:integer;
begin
     for k:=1 to n do
        for i:=1 to n do
           for j:=1 to n do
              if (a[i,k]<>-1) and (a[k,j]<>-1)
                 {exista drum de la i la k, si de la k la j}
                 and ( (a[i,j]=-1) or (a[i,k]+a[k,j] < a[i,j]) )
                 {si daca trecând prin k obtinem un cost mai mic}
              then a[i,j]:=a[i,k]+a[k,j] {retinem solutia mai buna}
end;
begin
     write('Dati numarul de varfuri: '); readln(n);
     for i:=1 to n do for j:=1 to n do a[i,j]:=-1;
     { Construim graful nul - fara nici un arc }
     writeln('Introduceti muchiile sub forma u v cost(uv):');
     writeln('Pentru a termina,introduceti 0 0 0.');
     readln(i,j,c)
     while i*j*c<>0 do begin
         if not(i in [1..n])or not(j in [1..n])or(i=j)or(c<=0) then
              writeln('Date eronata!')
         else a[i,j]:=c;
         readln(i,j,c);
     end;
     calculeaza_matricea_drumurilor_minime;
     repeat
          writeln('Dati varfurile intre care doriti costul minim:');
          writeln('Introduceti 0 0 pentru iesire:');
          repeat
                readln(i,j);
          until ((i=0) and (j=0))or((i in [1..n])and(j in [1..n]));
          if i<>0 then begin
             if a[i,j]=-1 then
                writeln('Intre ',i,' si ',j,' nu exista drum')
             else begin
                  writeln('Drumul minim are lungimea:',a[i,j]);
              {*} writeln
             end;
          end;
     until i=0;
end.
```


Scanned with OKEN Scanner

Găsirea efectivă a drumului de lungime minimă de la nodul *i* la nodul *j* se poate determina recursiv tot cu ajutorul tabloului *a* (al drumurilor minime) construit anterior:

```
procedure cauta(n1,n2:integer);
var i:integer;
begin
     for i:=1 to n do
        if (n1<>i)and(n2<>i)and(a[n1,i]+a[i,n2]=a[n1,n2]) then
            begin
                 cauta(n1,i);cauta(i,n2); exit
            end;
     write(n2,' ')
end;
```

Apelul procedurii se realizează în programul principal în locul marcat cu {*}: `writeln('Drumul minim=',i,' ');cauta(i,j);`

9.9 (X)

Fiind dat un graf orientat, să se determine toate componentele sale tari conexe.

R 9.9

Pe baza matricii drumurilor *d:*

$$d(i,j)=\begin{cases}1 & \text{dacă există drum de la nodul } i \text{ la nodul } j\\ 0 & \text{altfel}\end{cases}$$

calculată cu algoritmul Roy–Warshall se determină componentele tare conexe afişând fiecare nod care nu a mai fost afişat împreună cu toate nodurile care sunt şi succesori şi predecesori (din matricea drumurilor) ai nodului respectiv.

```
program tare_conexe;
const nmax=100;
type graf=array [1..nmax,1..nmax] of byte;
     marcaj=array [1..nmax] of boolean;
var a:graf;          {se initializeaza cu 0}
    i,j,k,n:integer;
    m:marcaj;        {se initializeaza cu false}
    modificare:boolean;
begin
     write('Dati numarul de noduri: ');
     readln(n);
     writeln('Dati arcele grafului (i j) : ');
     writeln('0 0 pentru sfarsit');
     readln(i,j);
     while i*j<>0 do begin
           a[i,j]:=1;
           readln(i,j)
     end;
     for k:=1 to n do                    {Roy-Warshall}
         for i:=1 to n do
             for j:=1 to n do
                 if a[i,j]=0 then
                    a[i,j]:=a[i,k]*a[k,j];
```


Scanned with OKEN Scanner

```
    k:=1;            {Determinarea componentelor tari conexe}
    for i:=1 to n do
        if not m[i] then begin
           write('Componenta tare conexa ',k,': ',i,' ');
           for j:=1 to n do
               if (j<>i) and (a[i,j]<>0) and (a[j,i]<>0) then begin
                  write(j,' ');m[j]:=true
               end;
           inc(k);writeln
        end
end.
```

Observaţie: Pentru aceeaşi problemă se putea utiliza un algoritm recursiv de marcare a tuturor predecesorilor şi succesorilor unui nod cu un acelaşi marcaj (plus – pentru succesori, minus – pentru predecesori).

```
.....
    plus,minus:array[1..max]of byte;   {se initializeaza cu 0}
procedure marcplus(i:byte);
var j:byte;
begin
     plus[i]:=k;
     for j:=1 to n do
         if (a[i,j]<>0)and(plus[j]=0) then marcplus(j)
end;
{ analog: procedure marcminus(i:byte);...}
begin...
     k:=1;
     for i:=1 to n do
         if plus[i]=0 then begin
            marcplus(i);marcminus(i);
            for j:=1 to n do
                if plus[j]<>minus[j] then
                   begin plus[j]:=0;minus[j]:=0 end;
                inc(k)
         end;
     for i:=1 to k-1 do begin
         write('Componenta conexa ',i,':');
         for j:=1 to n do
             if plus[j]=i then write(j,' ');
         writeln
     end
end.
```

9.10 (**XI – C**, Olimpiadă, faza judeţeană, Bistriţa Năsăud, 1995)

Se consideră un grup de n persoane, având fiecare un anumit număr de preocupări, cel mult p preocupări fiecare. Preocupările sunt codificate prin numere întregi nenule.

a) Să se determine perechile de persoane care au preocupări comune, specificând pentru fiecare astfel de pereche numărul de preocupări comune;

b) Să se determine perechea (eventual perechile) de persoane cu un număr maxim de preocupări comune;

c) Să se determine preocuparea îmbrăţişată de un număr maxim de persoane.

Scanned with OKEN Scanner

R 9.10

Problema propusă poate folosi ca model un multigraf, algoritmul fiind unul de numărare în care s-au memorat a preocupărilor fiecărei persoane ca mulţime de întregi. Pentru economie de memorie se puteau folosi liste (vezi capitolul Alocare dinamică), însă prelucrarea acestora ar fi fost mult mai greoaie.

```
program preocupari_comune;
type multime=set of byte;
var a:array[1..100]of multime;
    n,p,i,j,k,max,p1,p2:integer;
function card (x:multime):byte; {numarul de elemente ale multimii x}
var n,i:byte;
begin
     n:=0;
     for i:=1 to p do
         if i in x then inc(n);
     card:=n
end;
function frecventa(y:byte):byte;
 {frecventa de aparitie a preocuparii y}
var f,i:byte;
begin
     f:=0;
     for i:=1 to n do
         if y in a[i] then inc(f);
     frecventa:=f
end;
begin
  readln(n,p);
  for i:=1 to n do begin
      a[i]:=[];
      while not seekeoln do begin
            read(k);
            a[i]:=a[i]+[k];
      end;
      readln
  end;
  max:=0;
  p1:=0;p2:=0;
  writeln('Perechi cu preocupari comune:');
  for i:=1 to n-1 do
      for j:=i+1 to n do
          if a[i]*a[j]<>[] then begin
             write(i:3,j:3,': ');
             for k:=1 to p do
                 if k in a[i]*a[j] then write(k,' ');
             writeln;
             k:=card(a[i]*a[j]);
             if k>max then max:=k
          end;
  if max=0 then begin
     writeln('Nu exista preocupari comune');
     writeln('Toate preocuparile apar o singura data')
  end
```


Scanned with OKEN Scanner

```
  else begin
            writeln('Perechi cu cele mai multe preocupari comune');
            for i:=1 to n-1 do
                for j:=i+1 to n do
                    if card(a[i]*a[j])=max then writeln(i,' ',j);
            max:=0;
            write('Preocuparile cele mai intilnite: ');
            for i:=1 to p do begin
                k:=frecventa(i);
                if k>max then max:=k
            end;
            for i:=1 to p do if frecventa(i)=max then write(i,' ');
            writeln
  end
end.
```

Observaţie: pentru o rezolvare care să aibă în vedere toate situaţiile atragem atenţia că preocupările nu sunt neapărat codificate cu numere între 1 şi 255, iar valoarea lui p poate depăşi 255, deci tipul mulţime utilizat nu mai corespunde. Propunem realizarea unei numerotări a preocupărilor şi folosirea listelor de adiacenţă.

9.11 (X)

Într-o zonă de munte, există *n* cabane, între unele cabane existând trasee de legătură. Să se determine, dacă există, o ordine de vizitare a cabanelor, astfel încât să se parcurgă o singură dată toate traseele de legătură din zonă revenind la aceeaşi cabană de la care s-a pornit.

R 9.11

Modelul de rezolvare pentru problema enunţată este un graf neorientat în care se determină un ciclu eulerian (care să conţină toate muchiile grafului). În rezolvare se impune verificarea condiţiilor ca graful să fie eulerian: ignorând punctele izolate, graful trebuie să fie conex şi toate vârfurile să aibă grad par. În condiţiile problemei noastre devine absurdă ipoteza existenţei unor cabane izolate, deci nu ne rămâne decât să verificăm conexitatea şi faptul că de la fiecare cabană pleacă un număr par de trasee. Ordinea de vizitare se determină stabilind (cu ajutorul funcţiei *drum*) un ciclu, eliminând muchiile corespunzătoare acestuia şi recalculând gradele corespunzătoare vârfurilor sale şi apoi inserând în punctele cu grad neneul alte cicluri până ce se epuizează toate muchiile (*m*) ale grafului.

Am preferat, pentru memorarea ciclului eulerian, lucrul cu şiruri de caractere (în ipoteza că numărul muchiilor nu depăşeşte 255) pentru a realiza mai comod inserarea ciclurilor adiacente la cel existent. La fel de natural se lucra cu liste de numere şi, în ultimă instanţă cu tablouri unidimensionale simulând inserarea respectivă.

```
program graf_eulerian;
const max=16;
type tablou=array[1..max]of byte;
var a:array[1..max,1..max]of boolean;
    m:array[1..max] of boolean; {ambele se initializeaza cu false}
    gr:tablou;                  {se initializeaza cu 0}
    d:string; n,i,j,m:byte;
```


Scanned with OKEN Scanner

```
function conex:boolean; ...      {a se urmari problema 9.3}
function grade_pare:boolean;
begin
     grade_pare:=false;
     for i:=1 to n do begin
         for j:=1 to n do if a[i,j] then inc(gr[i]);
         if odd(gr[i]) then exit
     end;
     grade_pare:=true
end;
function drum(ini:byte):string;
var d:string;i,j:byte;
begin
     i:=ini;d:='';
     repeat
           j:=1;
           while not a[i,j] do inc(j);
           d:=d+chr(j);
           a[i,j]:=false;a[j,i]:=false;
           dec(gr[i]);dec(gr[j]);
           i:=j
     until i=ini;drum:=d
end;
begin
     write('Nr.cabane:');
     readln(n);
     writeln('Dati traseele i j (0 0 pentru sfarsit)');
     readln(i,j);
     m:=0;  {nr.muchii}
     while i<>0 do begin
           a[i,j]:=true;a[j,i]:=true;
           inc(m);readln(i,j)
     end;
     if conex and grade_pare then begin
        d:=chr(1);
        while length(d)<m do begin
              i:=1;
              while gr[ord(d[i])]=0 do inc(i);
              insert(drum(ord(d[i])),d,i+1)
        end;
        for i:=1 to m do write(ord(d[i]):3);
        writeln
     end
     else writeln('Nu este graf eulerian')
end.
```

9.12 (XI – C, Baraj pentru lotul lărgit, Slatina, 1995)

Fie un arbore cu n vârfuri. În fiecare nod al arborelui se află câte un bec. Prin atingerea unui bec acesta îşi schimbă starea (din aprins în stins şi invers), la fel şi vecinii lui direcţi. Considerând că iniţial toate becurile sunt stinse, să se scrie un program care generează o secvenţă de „atingeri" prin care pomul este aprins complet. Datele de intrare se citesc din fişierul *craciun.in* cu formatul: pe prima linie numărul de noduri, şi pe următoarele $n - 1$ linii muchiile arborelui.

Scanned with OKEN Scanner

R 9.12

Soluţia dată este de tip **backtracking**, având în vedere câteva observaţii:

– nu contează ordinea în care sunt făcute atingerile (cu alte cuvinte operaţiile sunt comutative);

– un nod trebuie atins cel mult o dată, deoarece două atingeri ale aceluiaşi vârf se anulează.

Folosind aceste considerente, vom alege o ordine în secvenţa de atingeri astfel încât să nu generăm o secvenţă de două ori. Concret, vârfurile atinse vor fi în ordine crescătoare.

Semnificaţia variabilelor folosite în program este următoarea:

n: numărul de noduri;

nra: numărul de noduri aprinse până la momentul curent;

a: matricea de adiacenţă a arborelui, în care pe diagonală se completează cu 1, pentru a uşura operaţiile făcute la marcarea unei atingeri;

b: un vector în care *b*[*i*] este 1 dacă becul din nodul *i* este aprins la momentul curent şi 0 dacă nu.

c: un vector în care *c*[*i*] = 1 dacă becul din nodul *i* a fost atins, 0 altfel.

Procedura recursivă *atinge* are ca parametru becul care trebuie atins. Ea încearcă pe rând să îl atingă sau să îl lase neatins, şi pentru fiecare caz se apelează pentru becul următor.

```
program craciun;
const nmax=100;
var i,j,k,n,nra:byte;
    a:array[1..nmax,1..nmax] of 0..1;
    b,c:array[1..nmax] of 0..1;
    f:text;
procedure afis;
var i:integer;
begin
  writeln('Au fost atinse urmatoarele becuri : ');
  for i:=1 to n do
      if c[i]=1 then write(i,' ');
  writeln
end;
procedure atinge(x:byte);
var i:byte;
begin
  if nra=n then begin
     afis;
     halt
  end;
  if x<=n then begin
    c[x]:=1;
    for i:=1 to n do
      if a[x,i]=1 then begin {marcare_atingere x}
         b[i]:=1-b[i]; {schimbarea starii becurilor vecine lui x}
         nra:=nra+b[i]*2-1; {actualizarea nr. de becuri aprinse}
      end;
    atinge(x+1);
```


Scanned with OKEN Scanner

```
    c[x]:=0;
    for i:=1 to n do
      if a[x,i]=1 then begin {stergere marcare_atingere x}
        b[i]:=1-b[i];
        nra:=nra+b[i]*2-1;
      end;
    atinge(x+1)
  end
end;
begin
  assign(f,'craciun.in');reset(f);
  readln(f,n);
  for i:=1 to n do for j:=1 to n do a[i,j]:=0;
  for i:=1 to n-1 do begin
    read(f,j,k);
    a[j,k]:=1;
    a[k,j]:=1;
  end;
  for i:=1 to n do a[i,i]:=1;
  nra:=0;
  atinge(1)
end.
```

După cum era de aşteptat programul este foarte lent, ordinul lui de complexitate fiind exponenţial. În continuare vom aduce o îmbunătăţire algoritmului, care nu-i va schimba ordinul de complexitate dar îl va face mult mai rapid pe majoritatea cazurilor uzuale. La programul anterior vom adăuga ca variabilă globală încă un vector *d* cu semnificaţia: *d*[*i*] este vârful cu indicele cel mai mare de care este legat direct vârful *i*. În aceste condiţii dacă s-a ajuns cu testarea la un vârf *x* care este mai mare decât un *d*[*i*] şi becul din vârful *i* nu este aprins, este inutil să continuăm cu căutarea deoarece *i* nu are vecini cu indici mai mari decât *x*, deci nu mai poate fi aprins în continuare. Astfel, la intrarea în procedura *atinge* se verifică dacă are sens să se continue sau nu. Variabila *nra* nu mai este necesară deoarece toate vârfurile au vecinii mai mici decât $n + 1$, deci când procedura se apelează pentru $n + 1$ se verifică practic dacă toate vârfurile sunt aprinse sau nu.

De remarcat că algoritmul acesta se comportă mai bine decât predecesorul său pentru arbori în care vârfurile sunt numerotate „ordonat", adică vecinii au asociate numere apropiate. Pentru următorul arbore particular (liniar), de exemplu, funcţionează aproape la fel de încet:

```
1--51--2--52--3--53--...--50--100,
```

dar pentru următorul merge mult mai repede:

```
1--2--3--4--5--...--100
```

```
program craciun2;
const nmax=200;
var i,j,k,n:byte;
    a:array[1..nmax,1..nmax] of 0..1;
    b,c:array[1..nmax] of 0..1;
    d:array[1..nmax] of 1..nmax;
    f:text;
```


Scanned with OKEN Scanner

```
procedure afis;
. . . . .
procedure atinge(x:byte);
var i:byte;v:boolean;
begin
  v:=true;
  for i:=1 to n do
      if (d[i]=x-1)and(b[i]=0) then v:=false;
  if v then begin
    if x>n then begin
       afis;
       halt
    end;
    c[x]:=1;
    for i:=1 to n do
        if a[x,i]=1 then b[i]:=1-b[i];
    atinge(x+1);
    c[x]:=0;
    for i:=1 to n do
        if a[x,i]=1 then b[i]:=1-b[i];
    atinge(x+1)
  end
end;
begin
  . . . . .
  for i:=1 to n do
      for j:=1 to n do
          if a[i,j]=1 then d[i]:=j;
  atinge(1)
end.
```

9.13 (Concurs ACM 1994, Bucureşti)

Să presupunem că un graf neorientat este format din n vârfuri şi m arce. Se ştie că sunt folosite toate muchiile pentru a lega vârfuri şi că două vârfuri pot fi legate direct prin cel mult o muchie. Să se determine valoarea maximă a expresiei $S=\sum_{i=1}^{n}\deg(i)^2$, unde $\deg(i)$ semnifică numărul de muchii legate de vârful i.

R 9.13

Prin $\deg(i)$ se înţelege deci gradul vârfului i. Propunem soluţia Greedy oferită de Bădoiu Mihai din Bucureşti; pentru verificare, se tipăreşte şi matricea de adiacenţă a grafului pentru care se atinge valoarea maximă a expresiei S:

```
program grad_max;
var v:array[1..100,1..100] of byte; {matricea de adiacenta}
    vs:array[1..100] of byte;       {veectorul gradelor}
    cost:integer;                   {cost = suma din enunt}
    n,m,i,j:integer;
procedure c2;{fixeaza acea muchie [x,y] care mmaximizeaza val.lui S}
var cc,x,y,i,j:integer;
begin
     cc:=cost;
```


Scanned with OKEN Scanner

```
    for j:=1 to n do
        for i:=1 to n do
            if (i<>j) and (v[i,j]=0) then
               if cc<cost+2*vs[i]+2*vs[j]+2 then begin
                  cc:=cost+2*vs[i]+2*vs[j]+2;
                  x:=i;
                  y:=j;
               end;
    inc(vs[x]); inc(vs[y]);
    v[x,y]:=1; v[y,x]:=1; cost:=cc;
end;
procedure calcul;
begin
    for i:=1 to m do c2;
end;
begin
    write('n , m = '); readln(n,m);
    if n*(n-1) div 2 < m then begin
       writeln('Prea multe arce'); halt
    end;
    for i:=1 to n do begin
        for j:=1 to n do v[i,j]:=0;
        vs[i]:=0
    end;
    cost:=0;
    calcul;
    writeln('cost = ',cost);
    for j:=1 to n do begin
        for i:=1 to n do write(v[j,i]:5);
        writeln
    end;
end.
```

Observaţie: dacă se fixează muchia [x,y], atunci valoarea sumei din enunţ creşte cu:

$$(\deg(x)+1)^2-(\deg(y)+1)^2-(\deg(x))^2-(\deg(y))^2=2\deg(x)+2(\deg(y)+2$$

9.14 (XI – C, Baraj studenţesc, 1994)

Într-un oraş intersecţiile sunt numerotate de la 1 la n (se consideră intersecţie nu numai locul unde se intersectează mai multe străzi ci şi capetele de străzi). Edilii oraşului doresc să numeroteze şi străzile oraşului, dar într-un mod care să ţină seama de numerotarea intersecţiilor, şi anume: două străzi diferite să aibă numere diferite şi în fiecare intersecţie trebuie să sosească o stradă care să aibă numărul intersecţiei. În condiţiile acestei probleme, prin *stradă* se înţelege o porţiune de drum aflată între două puncte de intersecţie, neglijând faptul că în practică străzile cuprind mai multe intersecţii. Întrebarea este dacă se poate – şi dacă da, cum se poate face o astfel de numerotare.

În fişierul de intrare se găsesc mai multe seturi de date separate de câte un rând liber; un set de date are pe prima linie numărul n ($n \le 150$) de puncte de intersecţie (terminale). Fiecare astfel de punct este identificat prin numărul cu care este numerotat, iar pe următoarele n linii sunt date pentru fiecare intersecţie k ($1 \le k \le n$) punctele cu care aceasta comunică direct printr-o stradă.

Scanned with OKEN Scanner

Pentru fiecare set de date trebuie afişată o numerotare a străzilor dacă există, respectiv mesajul: „Nu exista o astfel de numerotare" dacă nu există. Numerotarea se afişează astfel: pe fiecare linie o pereche de numere care reprezintă intersecţiile care delimitează strada şi apoi numărul străzii.

Exemplu:

Dacă un set de date din fişierul de intrare este:

```
4
2 3 4
1 3
2 1 4
3 1
```

ieşirea corespunzătoare este:

```
1 2 :  1
1 3 :  3
1 4 :  4
2 3 :  2
3 4 :  5
```

R 9.14.

Problema se poate reformula astfel: se dă un graf. Să se numeroteze muchiile sale, dacă este posibil, astfel încât în fiecare nod *i* să fie incidentă muchia cu numărul *i*.

Algoritmul de rezolvare pentru un set de date de intrare este următorul:

a) dacă există un nod de grad 0 nemarcat, atunci nu există soluţie;

b) dacă există un nod *i* de grad 1 nemarcat, muchia incidentă cu el va fi numerotată cu *i*, iar nodul va fi marcat;

c) dacă nu s-a ajuns în situaţiile precedente, se poate numerota orice muchie incidentă unui nod cu numărul nodului, iar nodul este marcat;

d) dacă toate nodurile au fost marcate, se numerotează muchiile care nu au fost încă numerotate.

La fiecare numerotare a unei muchii, gradele nodurilor cu care este incidentă scad cu 1.

```
program numerotare;
const max=150
var a,m:array[1..max,1..max]of byte;
    {m[i,j]=numerotarea muchiei [i,j]}
    grad,marc:array[1..max]of byte;
    q,i,j,k,l,n:integer;
    s:string; f,fo:text; b:boolean;

procedure solve;
  procedure num;
  begin
       j:=1;
       while (a[i,j]=0)and(j<=n) do inc(j);
       m[i,j]:=i;
       a[i,j]:=0; a[j,i]:=0;
       dec(grad[i]); dec(grad[j]);
       marc[i]:=1; b:=false
  end;
```


Scanned with OKEN Scanner

```
begin
     repeat b:=true; i:=1;
          repeat
                b:=true;
                if (grad[i]=0)and(marc[i]=0)then begin
                   writeln(fo,'Nu exista o astfel de numerotare');
                   writeln(fo); exit
                end;
                q:=i;
                for i:=1 to n do
                    if (grad[i]=1)and(marc[i]=0)and b then num;
                if b then begin
                   i:=q;
                   if (grad[i]<>0)and(marc[i]=0) then begin
                       num; i:=0
                   end;
                   inc(i)
                end
          until (i=n+1)or not b
     until b;
     k:=n+1;
     for i:=1 to n do
          for j:=1 to n do
              if a[i,j]=1 then begin
                 m[i,j]:=k; inc(k);
                 a[i,j]:=0;a[j,i]:=0
          end;
     for i:=1 to n do
          for j:=1 to n do
              if m[i,j]<>0 then writeln(fo,i,' ',j,': ',m[i,j]);
     writeln(fo)
end;
begin
     write('Dati numele fisierului de intrare '); readln(s);
     assign(f,s); reset(f);
     write('Dati numele fisierului de iesire '); readln(s);
     assign(fo,s); rewrite(fo);
     while not eof(f) do begin
           fillchar(grad,sizeof(grad),0);marc:=grad;
           fillchar(a,sizeof(a),0);m:=a;
           readln(f,n);
           for i:=1 to n do begin
               while not seekeoln(f) do begin
                     read(f,j);
                     a[i,j]:=1;a[j,i]:=1
                     inc(grad[i])
               end;
               readln(f)
           end;
           readln(f);
           solve
     end;
     close(f); close(fo)
end.
```

Observație: Soluția a fost dată de Stroe Mihai din București.

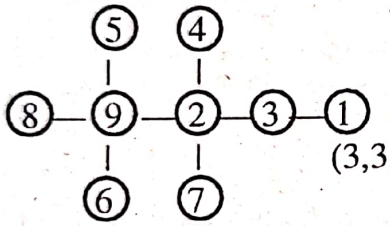
Scanned with OKEN Scanner

9.15 (**XI – C**, Concurs ACM 1993)

Se consideră un arbore cu n ($2 \le n \le 256$) noduri. Conform teoremei lui Cayley, numărul arborilor marcaţi este n^{n-2}. Prüfer a dat o demonstraţie constructivă petnru această teoremă, în care ideea principală este următoarea: se stabileşte o corespondenţă biunivocă între arborii marcaţi şi secvenţele de numere din mulţimea {1, 2, ..., n} având lungimea $n - 2$. Deoarece fiecare element al secvenţei poate fi orice număr din mulţime, rezultă că numărul arborilor este n^{n-2}. Mai jos este descrisă modalitatea prin care, plecând de la un arbore, se construieşte o secvenţă (unică) de lungime $n - 2$.

Se determină vârful final cu cel mai mic marcaj; fie acesta u. Fie $[u, v]$ muchia care leagă acest nod cu predecesorul său direct v. Eliminăm această muchie (adică se şterge vârful u împreună cu muchia $[u, v]$) şi introducem v ca următorul element al secvenţei. Procedeul se repetă până rămân numai două noduri. Metoda este ilustrată de următorul exemplu:

Pentru arborele:



obţinem secvenţele: (3), (3,3), (3,3,2), (3,3,2,2), (3,3,2,2,3), (3,3,2,2,3,2), (3,3,2,2,3,2,9)

În această problemă vom inversa procesul. Scopul este de a descoperi şi programa un algoritm care să construiască arborele plecând de la o secvenţă de lungime $n - 2$. Fişierul de intrare va conţine mai multe secvenţe. Fiecare secvenţă este dată sub forma unui şir de numere separate prin câte un blanc. Sfârşitul secvenţei este marcat cu 0, iar sfârşitul fişierului de intrare, cu – 1.

Ieşirea va da arborele sub forma unei liste de muchii, ca în exemplu.

Exemplu: Pentru fişierul de intrare:

```
3 3 2 2 3 2 9 0
-1
```

fişierul de ieşire va fi:

```
(1, 3) (4, 3) (5, 2) (6, 2) (7, 3) (3, 2) (2, 9) (8, 9)
```

R 9.15

Iniţial toate vârfurile care nu apar în secvenţa Prüfer dată (s) sunt disponibile. Pentru fiecare vârf i din s se determină primul vârf j (cel mai mic) disponibil. Se scrie muchia (i, j) declarând terminalul j indisponibil şi disponibilizând vârful i, dacă acesta nu mai apare pe poziţiile următoare ale lui s.

Observaţie: Algoritmul realizează funcţia inversă codificării Prüfer pentru arborii oarecare.

```
program acm; {reprezentarea unui graf}
uses crt;
var nn:word;{nr.noduri}
    fi,fo:text;nume_fi,nume_fo:string;  {Fisiere de intrare/iesire}
    x:integer;set_date:byte;
    e:array[1..256]of boolean;  {e[i]=false daca i este disponibil}
    s:array[1..256]of word;                    {secventa Prufer data}
```


Scanned with OKEN Scanner

```
 procedure err(a:string);                    {Tratarea erorilor}
   begin  sound(400);delay(200);nosound;
          writeln(a);writeln('Apasati o tasta');
          readkey;clrscr;halt
   end;
 procedure arbore;                          {Reconstituire arbore}
  var i,j:word; este:boolean;
  begin
       writeln(fo,'Set #',set_date,':');
       for i:=1 to nn-2 do if s[i]>nn then err('Date eronate');
       for i:=1 to nn do e[i]:=false;
       for i:=1 to nn-2 do e[s[i]]:=true;
       for i:=1 to nn-2 do begin
           j:=1; while e[j] do inc(j);
           writeln(fo,'(',j,',',s[i],')');
           e[j]:=true; este:=false;
           for j:=succ(i) to nn-2 do
               if s[i]=s[j] then este:=true;
           if not este then e[s[i]]:=false;
       end;
       e[s[nn-2]]:=true;
       j:=1;while e[j] do inc(j);
       writeln(fo,'(',s[nn-2],',',j,')')
  end;
begin
     write('Fisierul de intrare(CR pentru fisierul implicit):');
     readln(nume_fi);assign(fi,nume_fi);{$I-} reset(fi) {$I+};
     if ioresult<>0 then err('Nu exista fisierul de date');
     write('Fisierul de iesire (CR pentru fisierul implicit):');
     readln(nume_fo);assign(fo,nume_fo);rewrite(fo);
     if nume_fi='' then writeln('Dati un set de date sau -1');
     {$I-}read(fi,x){$I+};set_date:=0;
     if ioresult<>0 then err('Fisier vid sau format incorect');
     while x<>-1 do begin
           nn:=0;
           while x<>0 do begin
                 inc(nn);s[nn]:=x;read(fi,x)
           end;
           nn:=nn+2;
           readln(fi);
           inc(set_date);
           arbore;
           if nume_fi='' then writeln('Dati un set de date sau -1');
           read(fi,x)
     end;
     close(fi);close(fo);
     writeln('Gata');readln
end.
```

9.16 (X – C, Pregătire Timişoara)

Dându-se n numere pozitive $d_1, d_2, .., d_n$ să se verifice dacă există un arbore cu n noduri ale căror grade sunt $d_1, d_2, .., d_n$. Dacă există un astfel de arbore, să se reprezinte prin liste ale nodurilor sale. O listă a unui nod conţine numărul nodului urmat de fiii săi.

Scanned with OKEN Scanner

Exemplu: Pentru intrarea:

```
1 1 2 3 1 1 3
```

o ieşire corectă este:

```
DA
1 2 3 4, 2 5 6, 3 7, 4, 5, 6, 7
```

R 9.16

Se sortează gradele nodurilor, apoi se aplică următorul algoritm:

– se porneşte de la arborele cu rădăcina 1 şi cu $d[1]$ frunze (dacă este posibil);

– cât timp nu s-au introdus n noduri în arbore, se înlocuieşte, dacă este posibil, prima frunză cu primul nod i neintrodus încă în arbore; acest nod va avea $d[i] - 1$ fii;

– dacă s-a reuşit introducerea celor n noduri, s-a ajuns la soluţie care se tipăreşte; altfel, se afişează un mesaj.

Observaţie: Soluţia nu este neapărat unica.

```
var fi,fo:text;s:string;
    d:array[1..200]of byte;
    vecini:array[1..200,1..200]of byte;
    i,j,k,l,m,n:byte;
begin
  assign(fi,'int10.txt'); reset(fi);
  n:=0;
  while not eoln(fi)do begin
        inc(n);
        read(fi,d[n]);
  end;
  close(fi);
  for i:=2 to n do
      for j:=1 to i-1 do
          if d[i]>d[j]then begin
             k:=d[i];
             d[i]:=d[j];
             d[j]:=k;
          end;
  for i:=2 to d[1]+1 do begin
      vecini[i,1]:=1;
      vecini[1,i]:=1;
  end;
  k:=d[1]+2;
  for i:=2 to n do begin
      if (d[i]-2<=n-k)and(i<k) then
         for j:=1 to d[i]-1 do begin
             vecini[i,k]:=1;
             vecini[k,i]:=1;
             inc(k)
         end
      else begin
          assign(fo,'out10.txt'); rewrite(fo);
          writeln(fo,'Imposibil');close(fo);halt
      end
 end;
 writeln;
```


Scanned with OKEN Scanner

```
  assign(fo,'out10.txt'); rewrite(fo);
  for i:=1 to n do begin
      if d[i]>1 then write(fo,i,':') else write(fo,i,' ');
      for j:=i+1 to n do
          if vecini[i,j]=1 then write(fo,j,' ');
      if i<>n then write(fo,', ');
  end;
  close(fo)
end.
```

Observaţii: Soluţia a fost dată de Stroe Mihai din Bucureşti.

Vă recomandăm să studiaţi şi teorema Hakimi pentru secvenţe grafice ([4]).

9.17 (XI)

Se dă un graf $G = (V, E)$ şi se cere să se găsească o mulţime intern stabilă de vârfuri cu proprietatea că este de cardinal maxim cu această proprietate.

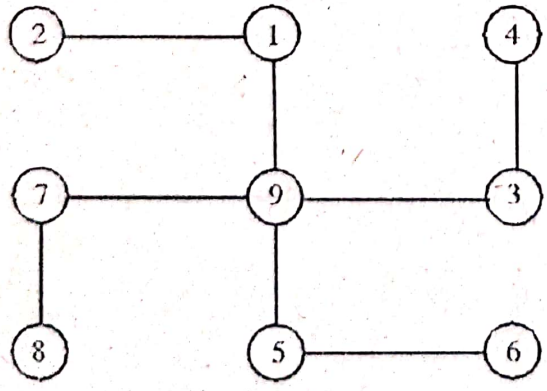
R 9.17

O mulţime S de vârfuri cu proprietatea că oricare două vârfuri distincte sunt neadiacente formează o mulţime intern stabilă. Se poate demonstra că această problemă este NP-completă, adică nu există algoritmi polinomiali pentru rezolvarea ei. Totuşi se poate folosi următorul algoritm euristic:

1. Se defineşte ca mulţime iniţială mulţimea $V' = V$ a tuturor vârfurilor;
2. Se verifică dacă V' este mulţime intern stabilă. Dacă da, se sare la pasul 5;
3. Se elimină din V' vârful care are cele mai multe legături cu celelalte vârfuri din V'. Dacă sunt mai multe cu acelaşi număr (maxim) de legături, se elimină unul dintre ele;
4. Se trece la pasul 2;
5. V' este mulţimea căutată.

Se observă imediat că algoritmul are complexitate polinomială, dar, deşi el merge pe majoritatea cazurilor, există situaţii în care nu oferă soluţia optimă.

Un contraexemplu clasic pentru acest algoritm ar fi 'svastica':



Aici:

$G = (\{1, .., 9\}, \{\{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8\}, \{1, 9\}, \{3, 9\}, \{7, 9\}, \{5, 9\}\})$. Luăm iniţial $V' = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Eliminăm 9, iar apoi (să zicem) 1, 3, 5, 7. Vom obţine: mulţimea intern stabilă $V' = \{2, 4, 6, 8\}$. Se observă însă că ea nu este de cardinal maximal cu această proprietate; mulţimea $V'' = \{2, 4, 6, 8, 9\}$ este tot o mulţime intern stabilă, şi are mai multe elemente decât V'.

Totuşi, aplicarea acestui algoritm se recomandă atunci când datele de intrare sunt mari şi nu este neapărat necesară o soluţie optimă, ci una cât mai bună. Vom

Scanned with OKEN Scanner

prezenta programul bazat pe algoritmul descris mai sus. Programarea unui algoritm corect pentru aflarea mulţimii maxime intern stabile (prin **backtracking**) o lăsăm ca exerciţiu.

```
program intern_stabila;
const
     nmax=100;
var
    a:array [1..nmax,1..nmax] of boolean;{se initializeaza cu false}
    v:array[1..nmax] of boolean;
    v1:array[1..nmax] of integer;
    max,i,j,n,c:integer;
procedure calc_gr;  { Calculeaza nr de legaturi cu varfuri din V'}
var i,j:integer;
begin
     for i:=1 to n do
         if v[i] then begin
            v1[i]:=0;
            for j:=1 to n do
                if v[j] and a[i,j] then inc(v1[i]);
         end;
end;
function ver:boolean; {Verifica daca V' este intern stabila}
var i:integer;
begin
  for i:=1 to n do
    if v[i] and (v1[i]<>0) then begin
      ver:=false;
      exit
    end;
  ver:=true;
end;
begin
     write('Dati numarul de varfuri: ');
     readln(n);
     writeln('Dati muchiile sub forma u v (0 0 pentru iesire) :');
     .... {secventa de citire}
     for i:=1 to n do
         v[i]:=true; {multimea V' }
     calc_gr;
     while not ver do begin
           max:=0;
           for i:=1 to n do
               if v[i] then if v1[i]>max then max:=v1[i];
           {Calculeaza nr.de legaturi ale varfului ce se elimina}
           for i:=1 to n do if v[i]and(v1[i]=max) then begin
                 v[i]:=false; break   {Elimina varful respectiv}
           end;
           calc_gr
     end;
     writeln('Multimea maxima intern stabila este:');
     for i:=1 to n do
         if v[i] then write(i,',');
     writeln(#8,'.')
end.
```


Scanned with OKEN Scanner

Probleme propuse

9.18 (X)

Imperiul spaţial Mac este format din n civilizaţii. Există însă perechi de astfel de civilizaţii între care nu domneşte înţelegerea. La o reorganizare a lor s-a hotărât ca acestea să fie împărţite în cât mai puţine federaţii, astfel încât în fiecare federaţie să nu existe divergenţe. Împărţiţi imperiul în federaţii.

Observaţie: Dându-se un graf neorientat , să se coloreze cu număr minim de culori fiecare vârf al său, astfel încât oricare două vârfuri legate printr-o muchie să fie colorate diferit. Fiecare civilizaţie este un vârf al grafului, iar civilizaţiile între care există divergenţe vor fi unite prin câte o muchie. O federaţie este formată din toate vârfurile cu aceeaşi colorare. Vă recomandăm să detectaţi echivalenţa acestei probleme cu problema 9.5.

9.18 (X – C)

Un principiu al organizării unei reţele de informaţii este principiul siguranţei (prin care informaţia se poate propaga între oricare două puncte ale reţelei, chiar în ipoteza că o singură legătură directă este întreruptă accidental). Cunoscându-se numărul nodurilor reţelei (codificate prin numere naturale între 1 şi n, n dat) şi legăturile directe informaţionale existente, să se determine reţele informaţionale cu un număr maxim de noduri care se pot forma.

Observaţie: Se cere de fapt să se găsească toate componentele biconexe ale unui digraf.

9.19 (X)

Cunoscându-se că într-un grup de n persoane codificate prin numere între 1 şi n, fiecare persoană are o listă (dată) de alte persoane pe care le va informa deîndată ce află un anumit mesaj, să se determine dacă există persoane care vor primi de cel puţin două ori acelaşi mesaj.

9.20 (X – C)

Se dă un graf neorientat, prin matricea existenţei drumurilor. Să se orienteze muchiile grafului astfel încât graful nou obţinut – orientat – să fie:

a) tare conex;

b) unilateral conex.

9.21 (X – C)

Se dau N cărămizi (prin dimensiunile lor). Se cere să se construiască un turn format din cât mai multe cărămizi. O cărămidă poate fi aşezată peste alta în orice poziţie, cu condiţia ca baza ei să se sprijine în totalitate pe cărămida precedentă (dreptunghiul care formează baza să fie inclus strict în dreptunghiul care formează baza precedentei cărămizi). Pentru simplificare, se consideră că toate cărămizile care formează turnul au laturile paralele.

Indicaţie: Se construieşte un digraf în care se asociază fiecărei cărămizi câte trei vârfuri (reprezentând feţe neparalele ale aceleiaşi cărămizi), şi se pun arce între vârfurile care reprezintă feţe ce pot fi suprapuse. În graful astfel creat se caută un drum de lungime maximă (graful neavând cicluri, i se poate aplica algoritmul Dijkstra sau Floyd).

Scanned with OKEN Scanner

9.22 (X)

Doi prieteni, absolvenţi ai Institutului Agronomic din Bucureşti doresc să îşi caute loc de muncă la ferme diferite, dar în aşa fel încât distanţa dintre ferme să fie cât mai mică iar legătura dintre ferme să se facă printr-o altă fermă (una cu birt!). Cunoscând numărul n de ferme care oferă locuri de muncă şi m distanţe dintre unele ferme (cele care au cale de acces pietruită), să se afişeze numerele de ordine ale perechilor de ferme unde-şi pot alege cei doi prieteni locul de muncă.

Indicaţie: Fermele vor fi aşezate în vârfurile unui graf în care distanţele sunt muchiile grafului. Ceea ce ne interesează pe noi (de fapt pe cei doi tineri!) este determinarea drumurilor minime de lungime 2 (formate din două muchii). Drumurile de lungime 2 se obţin calculând A^2, unde A este matricea de adiacenţă (amintim că se efectuează înmulţirea booleană a lui A cu ea însăşi). $A^2[i, j] = 1$ implică faptul că între nodurile i şi j există drum de lungime 2 şi nu ne rămâne decât să determinăm care dintre perechile (i, j) au cale de acces minimă prin intermediul altei ferme k.

9.23 (X)

Se consideră afinitate lexicală între două cuvinte o relaţie prin care fiecare dintre cuvinte începe cu o literă care se găseşte şi în celălalt cuvânt. Dacă se citesc de la tastatură n cuvinte formate numai din litere majuscule după forma din exemplu, să se afişeze cel mai numeros grup de cuvinte cu proprietatea că între oricare două dintre ele există afinitate lexicală.

Indicaţie: Am putea încerca şi o rezolvare în care să construim mulţimi de indici ai cuvintelor între care există afinitate lexicală, pornind de la premisa optimistă că nu se furnizează mai mult de 256 cuvinte şi apoi să determinăm mulţimea de cardinal maxim. Vom propune însă o metodă care construieşte graful asociat cuvintelor şi relaţiilor dintre ele, graf memorat prin matricea sa de adiacenţă. Cerinţa problemei este echivalentă cu determinarea subgrafului complet cu număr maxim de noduri.

Scanned with OKEN Scanner

X. PROGRAMARE DINAMICĂ

Introducere

Datorită faptului că este una dintre cele mai complexe tehnici de programare, vom acorda o atenţie mărită acestei modalităţi de tratare a problemelor, atenţie însoţită de un număr mare de exemple. Ele sunt în majoritate de facturi diferite, şi constituie un bun punct de pornire în formarea unei gândiri dinamice.

Datorită frecvenţei folosirii aceluiaşi termen, vom nota noţiunea de programare dinamică prin iniţialele **PD**.

PD este o tehnică de programare de nivel înalt în sensul că necesită o îmbinare între intuiţie şi rigurozitate matematică. Expunem, într-un mod intuitiv, ideile acestei tehnici.

În general, termenul de „principiul optimalităţii" semnifică următoarele: atunci când vrem să obţinem o situaţie favorabilă dintr-un anumit punct de vedere, este bine ca ea să rezulte tot din situaţii favorabile. Un exemplu intuitiv este acela al distanţelor:

Într-un cartier sunt mai multe case, pe care le vom nota cu litere: A, .., Z. Dacă există un drum minim de la A la Z şi acest drum trece de exemplu prin D, atunci porţiunile din acest drum situate între A şi D, respectiv D şi Z sunt minime.

Pentru a justifica aceasta, să presupunem prin absurd că drumul A-α-D nu ar fi minim. Înseamnă că există alt traseu A-β-D mai scurt decât acesta; dacă luăm atunci drumul A-β-D-Z, el va fi evident mai scurt decât drumul A-α-D-Z, care este considerat ca fiind drum minim, contradicţie.

Deci, dacă traseul A-Z este optim, atunci orice subtraseu al său este optim.

Principiul optimalităţii are trei variante:

1. Dacă $s_1, s_2, ..., s_n$ este un şir optim de transformări care duc un sistem din starea iniţială în starea finală, atunci pentru orice i ($1 \le i \le n$), $s_1, s_2, ..., s_i$ este un şir optim de transformări. Deci la fiecare moment i se alege o transformare astfel încât şirul $s_1, s_2, ..., s_i$ să fie optim.

2. Dacă $s_1, s_2, ..., s_n$ este un şir optim de transformări care duc sistemul din starea iniţială în starea finală, atunci pentru orice i ($1 \le i \le n$), $s_i, s_{i+1}, s_{i+2}, ..., s_n$ este un şir optim de transformări. Deci la fiecare pas i se alege acea transformare pentru care $s_i, s_{i+1}, ..., s_n$ este un şir optim de transformări.

3. Dacă $s_1, s_2, ..., s_i, s_{i+1}, ..., s_n$ este un şir optim de transformări care duc sistemul din starea iniţială în starea finală, atunci pentru orice i ($1 \le i \le n$), $s_1, s_2, ..., s_i$ şi $s_i, s_{i+1}, ..., s_n$ sunt două şiruri optime de transformări. Deci vom încerca la fiecare pas să alegem două subşiruri optime care alăturate să ducă la un şir optim.

De precizat că reciproca nu este adevărată totdeauna: din îmbinarea a două optime nu rezultă neapărat un optim; doar dacă avem un optim, putem fi siguri că el este compus din două optime. Deci trebuie să facem toate combinaţiile de soluţii parţiale optime pentru a găsi acea pereche de optime care alăturate conduc la un optim. Am folosit mai sus termenul de optim, dar nu am precizat în ce sens. Criteriul ales nu este valabil pentru orice problemă, ci va fi un criteriu specific problemei pe care o tratăm. În exemplul anterior criteriul era distanţa dintre două puncte (de exemplu A şi Z). Deci noţiunea abstractă cu care se lucra în problemă era cea de distanţă, iar optim însemna lungimea ei, măsurată în anumite unităţi.

Dacă – spre exemplu – vom aborda o problemă care necesită să evaluăm ce preţ trebuie plătit pentru a duce un sistem dintr-o stare în alta, noţiunea cu care se va lucra este

Scanned with OKEN Scanner

banul, iar optim va însemna cât mai puţini. După cum se poate intui, întotdeauna se va găsi o cuantificare a realităţii astfel încât optimalitatea să se refere la numere.

Rezolvările folosind **PD** sunt în general bazate pe realizarea unei optimizări, de aflare a unui şir optim de transformări conform criteriului care ne interesează.

Deficienţa acestei metode este că depinde de experienţa personală şi de capacitatea de a intui. Şi încă un lucru: să nu vă speriaţi; de obicei programele de **PD** sunt extrem de scurte şi neaşteptat de simple.

Probleme rezolvate

10.1 (VI)

Un om doreşte să urce o scară cu N trepte (N dat la tastatură). El poate urca una sau două trepte la un moment. Fiind o fire curioasă, omul nostru îşi pune problema în câte moduri poate urca această scară.

R 10.1 (Ordinul de complexitate al algoritmului: N)

```
program scara;
const nmax=100;
var   i,n:integer;
      a:array [1..nmax] of longint;
begin
     write('n=? ');
     readln(n);
     a[1]:=1;
     a[2]:=2;
     for i:=3 to n do
        a[i]:=a[i-1]+a[i-2];
     writeln('Omul poate urca scara in ',a[n],' moduri.')
end.
```

Acesta este unul din puţinele exemple în care optimul nu este foarte vizibil.

Să gândim intuitiv în felul următor: suntem pe treapta i, nu are importanţă care este aceea. Ca să ajungem aici, înainte am călcat ori pe treapta $i - 1$ şi de acolo am făcut un pas de o treaptă, ori am călcat pe treapta $i - 2$ şi am făcut un pas de două trepte. Ce scrie mai sus este adevărat pentru că omul nu poate face decât două *transformări*: un pas de o treaptă sau un pas de două trepte.

Deci să notăm cu A[i] numărul de moduri în care se poate ajunge la treapta i, oricare ar fi aceasta. Acest număr este egal cu numărul de moduri de a ajunge la treapta $i - 1$ (adică A[$i - 1$]) plus numărul de moduri de a ajunge la $i - 2$ (adică A[$i - 2$]). Am numărat deja toate variantele în care se poate ajunge la treapta i: A[i] = A[$i - 1$] + A[$i - 2$].

Pe prima treaptă putem ajunge într-un singur fel, cu un pas simplu: A[1] = 1. Pe a doua treaptă putem ajunge în două feluri: un pas simplu urmat de alt pas simplu, sau un pas dublu, deci A[2] = 2. De la treapta a treia încolo folosim relaţia A[i] = A[$i - 1$] + A[$i - 2$]. După cum se vede, acesta este de fapt şirul Fibonacci în una din aplicaţiile sale „drăguţe".

Scanned with OKEN Scanner

10.2 (VIII)

Se dă un şir de numere A[1], A[2], ..., A[n] cu n dat la tastatură. Să se afle cel mai lung subşir crescător.

R 10.2 (Ordin de complexitate N^2)

```
program subsir_crescator;
const nmax=100;
var   a,b,c:array [1..nmax] of integer;
      i,j,n:integer;
begin
  write('n='); readln(n);
  writeln('Dati pe linia urmatoare sirul dvs.');
  for i:=1 to n do read(a[i]);
  readln;
  b[n]:=1; c[n]:=0;
  for i:=n-1 downto 1 do
      begin
           c[i]:=0; b[i]:=1;
           for j:=i+1 to n do
               if (a[i]<a[j]) and (b[i]<b[j]+1) then
                  begin
                       b[i]:=b[j]+1;
                       c[i]:=j
                  end
      end;
  write('Sirul este:');
  j:=1;
  for i:=2 to n do
    if b[i]>b[j] then j:=i;
  while c[j]<>0 do
        begin
             write(' ',a[j]);
             j:=c[j]
        end;
  write(' ',a[j]);
  writeln
end.
```

Şirul nostru este A[i], cu $1 \le i \le N$. În problemă ne interesează cel mai lung subşir crescător, deci noţiunea abstractă cu care lucrăm este lungimea unui subşir, iar optimalitatea se referă la o lungime cât mai mare. Deci vom alege o reprezentare a datelor în care să se reflecte direct sau indirect noţiunea de lungime şi aceea de lungime maximă. Definim şirul B[i], cu $i \in [1, N]$. B[i] reprezintă lungimea maximă a subşirului care se obţine dacă se alege A[i] ca prim element al subşirului soluţie. Odată luat elementul A[i] ca prim element al şirului soluţie, este clar că toate elementele mai mari decât el, care se află în dreapta sa, pot fi folosite. Acest lucru se reflectă în şirul B[i] : B[i] = 1 înseamnă că elementul A[i] este luat ca prim element, şi presupunem că el este şi ultimul. Vedem dacă există un element mai mare în dreapta sa, împreună cu care lungimea unui nou şir să fie

Scanned with OKEN Scanner

maximă, adică vom căuta acel $j > i$ (dacă există) astfel încât A[j] > A[i], adică poate fi pus în dreapta lui A[i], şi lungimea noului şir este mai mare decât cea a unui şir pe care îl avem până acum, adică B[i] < B[j] + 1. Operaţia + 1 de mai înainte arată că lungimea noului şir A[i], A[j], ... creşte cu 1 prin punerea lui A[i]. Pentru a putea reconstitui apoi subşirul soluţie, în C[i] reţinem ultimul j care dă o soluţie optimă, sau 0 dacă nu mai sunt elemente în dreapta lui A[i] care să intre în subşirul soluţie. Aici este aplicat principiul optimalităţii în varianta *înainte*, când se caută un element soluţie în faţa unui eventual subşir optim.

10.3 (IX)

Se dau două şiruri A[1], A[2], ..., A[n] şi B[1], B[2], ..., B[m]. Să se găsească cel mai lung subşir comun al acestora.

R 10.3 (Ordin de complexitate al algoritmului: N^2)

```
program subsir_comun;
const nmax=80;
var   a,b,s:array [1..nmax] of integer;
      c:array [1..nmax,1..nmax] of integer;
      i,j,is,js,ls,n,m:integer;

procedure calc(i,j:integer);
var i1,j1,k:integer;
begin
     c[i,j]:=0;
     if a[i]<>b[j] then exit; { prin conventie }
     k:=0;
     for i1:=1 to i-1 do
         for j1:=1 to j-1 do
             if c[i1,j1] > k then
                k:=c[i1,j1];
     c[i,j]:=k+1;
     if c[i,j] > ls then
        begin
             is:=i;
             js:=j;
             ls:=c[i,j]
        end
end;

procedure gaseste_subsir(i,j,nr_el:integer);
 var i1,j1:integer;
begin
     s[nr_el]:=a[i];
     if c[i,j]>0 then
        for i1:=1 to i-1 do
            for j1:=1 to j-1 do
                if c[i1,j1] = nr_el-1 then
                   begin
                        gaseste_subsir(i1,j1,nr_el-1);
                        exit
                   end
end;
```


Scanned with OKEN Scanner

```
begin
      write('n=');
      readln(n);
      writeln('Dati pe linia urmatoare sirul A:');
      for i:=1 to n do read(a[i]);
      readln;
      write('m='); readln(m);
      writeln('Dati pe linia urmatoare sirul B:');
      for i:=1 to m do read(b[i]);
      readln;
      ls:=0;
      for i:=1 to n do
          for j:=1 to m do calc(i,j);
      if ls>0 then gaseste_subsir(is,js,ls);
      writeln('Cel mai lung subsir comun are ',ls,' el. si este:');
      for i:=1 to ls do write(' ',s[i]);
      writeln
end.
```

Definim matricea C[*i*, *j*] cu $1 \le i \le N$, $1 \le j \le M$ cu semnificaţia C[*i*, *j*] = 0 dacă A[*i*] ≠ B[*j*], iar dacă A[*i*] = B[*j*] atunci C[*i*, *j*] = lungimea celui mai lung subşir comun din care face parte şi elementul A[*i*] (sau B[*j*]). Algoritmul de rezolvare este conform principiului optimalităţii varianta *înapoi*, când există un eventual şir de transformări optime şi adăugăm elemente în coada sa.

Odată definită convenţia de mai sus, rezolvarea este simplă: pentru orice A[*i*] = B[*j*] care face deci parte din soluţie, căutam în primele *i* – 1 elemente din şirul A şi primele *j* – 1 elemente din al şirului B, cel mai lung subşir comun (sau şirul vid, dacă nu există ceva), la care practic adaugăm elementul curent.

În final se caută în matrice elementul C[*i*, *j*] maxim, adică şirul format cu elemente care se termină cu A[*i*] (sau B[*j*] identic) este maxim; procedeul se reia pe matricea formată din primele *i* – 1 linii şi primele *j* – 1 coloane din matricea C.

10.4 (XI – C, [2])

Avem o tablă de şah, de dimensiune standard 8 × 8, pe care se află diverse piese de şah. Un cal se află pe poziţia (*i*, *j*). Dorim să mutăm calul în poziţia (*k*, *l*) – dacă acest lucru este posibil – folosind oricare din cele 8 mutări posibile ale calului în orice succesiune, astfel încât numărul de mutări să fie minim.

Intrarea este dată dintr-o matrice de caractere de forma

```
*-**-***
**-***-*
*****-**
***-****
********
******-*
********
********
```

unde "*" înseamnă poziţie ocupată, iar "-" poziţie liberă.

R 10.4

```
const d:array[1..8]of integer=(1,1,-1,-1,2,2,-2,-2);
      s:array[1..8]of integer=(2,-2,2,-2,1,-1,1,-1);
```


Scanned with OKEN Scanner

```
var  t:array [1..8,1..8] of char;
     a:array [1..64,1..64] of integer;
     n,i,j,k,l,ic,jc,kc,lc:integer;
function poate_sari(i,j:integer;
var k,l:integer;ai,aj:integer):boolean;
begin
     k:=i-ai; l:=j-aj;
     if (k in [1..8])and(l in [1..8])and(t[k,l]='-')
        then poate_sari:=true
        else poate_sari:=false
end;
procedure adauga_muchie(i,j,k,l:integer);
begin
     a[(i-1)*8+j,(k-1)*8+l]:=1;
     a[(k-1)*8+l,(i-1)*8+j]:=1
end;
procedure calculeaza_matricea_drumurilor_minime;
 var k,i,j:integer;
begin
  for k:=1 to 64 do
    for i:=1 to 64 do
      for j:=1 to 64 do
        if (a[i,k]>0)and(a[k,j]>0)and
         ((a[i,j]=0)or (a[i,k]+a[k,j]<a[i,j]) )
         then a[i,j]:=a[i,k]+a[k,j]
end;

begin
     writeln('Dati pe liniile urmatoare configuratia tablei:');
     for i:=1 to 8 do begin
         for j:=1 to 8 do read(t[i,j]); readln
     end;
     writeln; writeln('Tabla este:  ');
     for i:=1 to 8 do begin
         for j:=1 to 8 do write(t[i,j]); writeln
     end;
     write('Pozitia initiala:');readln(ic,jc);
     write('Pozitia finala  :');readln(kc,lc);
     if (t[ic,jc]='*') or (t[kc,lc]='*') then begin
        writeln('Deplasare imposibila !'); exit;
     end;
     fillchar(a,sizeof(a),0);
     for i:=1 to 8 do
         for j:=1 to 8 do
             if t[i,j]='-' then begin
                for k:=1 to 8 do
                    if poate_sari(i,j,k,l,d[k],s[k])then
                       adauga_muchie(i,j,k,l)
             end;
     calculeaza_matricea_drumurilor_minime;
     if a[(ic-1)*8+jc,(kc-1)*8+lc]=0 then
        writeln('Deplasare imposibila !')
     else begin
          writeln('Se fac ',a[(ic-1)*8+jc,(kc-1)*8+lc],' mutari');
          writeln('Din pozitia (',ic,',',jc,') in (',kc,',',lc,')')
     end
end.
```


Scanned with OKEN Scanner

O altă metodă de rezolvare ar fi **backtracking**. Inconvenientul este că am calcula atunci un drum de mai multe ori, ceea ce contravine logicii pure. S-ar putea aborda şi metoda **branch & bound** care complică de asemenea calculele şi duce la mărirea ineficientă a dimensiunii programului. Acesta este genul de problemă la care se poate aplica matricea drumurilor minime, prin PD.

Algoritmul este de fapt acelaşi ca la matricea drumurilor minime, cu deosebirea că graful nu se dă, ci îl construim noi. Avem aici o tehnică de programare auxiliară în sensul că permite aducerea unei probleme la una cunoscută, pe care o numim „liniarizarea de matrici".

Iniţial am avut o problemă în care era dată o matrice de dimensiuni 8×8 care simboliza ocuparea sau neocuparea unor poziţii pe tabla de şah. „Tehnica" de liniarizare consta în a considera fiecare element al matricii ca fiind un nod al unui graf.

De exemplu elementul $A[i, j]$ al matricii va deveni nodul $(i - 1) * 8 + j$, unde 8 este numărul de coloane ale matricii. Matricea este de forma:

```
A[1,1] A[1,2] .... A[1,8]
A[2,1] A[2,2] .... A[2,8]
...
A[8,1] A[8,2] .... A[8,8]
```

Este ca şi cum am considera fiecare element al matricii ca fiind o entitate, (nod) dar pe rând: prima linie primul element, prima linie al doilea element, .., a doua linie primul element, ... şi le-am numerota în ordine crescătoare începând cu 1. Corespondenţa între elementul de coordonate (i, j) şi nodul $(i - 1) * 8 + j$ unde i este numărul liniei, iar j numărul coloanei, se poate verifica foarte uşor că este biunivocă.

Odată considerat un graf simplu, putem construi matricea drumurilor minime, lucru care în exemplul nostru este notat prin $A[i, j]$ cu $1 \le i \le 64$, $1 \le j \le 64$. În acest graf vom construi o muchie între nodul $(i - 1)* 8 + j$ şi $(k - 1) * 8 + l$ dacă poziţiile (i, j) respectiv (k, l) de pe tabla noastră sunt libere şi se poate ajunge prin săritură directă de la (i, j) la (k, l). Ponderea arcului pe care îl facem va fi 1, exact numărul de sărituri pe care le facèm pentru deplasare între cele două poziţii.

După ce calculăm matricea drumurilor minime, $A[(i - 1)*8 + j, (k - 1)*8 + l]$ va arăta exact numărul minim de sărituri ale calului pentru a ajunge din poziţia (i, j) în poziţia (k, l).

Atragem atenţia că multe probleme se pot rezolva prin liniarizări de matrici, lucru care impune o atenţie mărită acestei metode. Ca un dezavantaj, în unele probleme, se poate dori liniarizarea unei matrici foarte mari, deci metoda necesită mult spaţiu. Deoarece însă tendinţele actuale se îndreaptă spre micşorarea timpului de rulare al unui program, sfatul nostru este acela că întotdeauna între un algoritm care necesită mult spaţiu şi unul care necesită mult timp de rulare, să îl preferaţi pe cel care va funcţiona mai rapid.

10.5 (XI)

Să se rezolve problema 10.4 fără a folosi liniarizare de matrici.

Scanned with OKEN Scanner

R 10.5 (Ordin de complexitate al algoritmului: N^2)

```
var t:array [1..8,1..8] of char;
    a:array [1..8,1..8] of integer;
    pas,i,j,k,l,ic,jc,kc,lc:integer;
    modificare:boolean; fi:text;

function poate_sari(i,j:integer;
 var k,l:integer;ai,aj:integer):boolean;
begin
     k:=i-ai; l:=j-aj;
     if (k in [1..8])and(l in [1..8])and(t[k,l]='-')and(a[k,l]>pas)
        then poate_sari:=true
        else poate_sari:=false
end;

procedure marcheaza(i,j:integer);
begin
     a[i,j]:=pas+1;
     modificare:=true;
end;

begin
     writeln('Va rugam sa dati pe liniile configuratia tablei:');
     for i:=1 to 8 do begin
         for j:=1 to 8 do read(t[i,j]); readln;
     end; writeln;
     write('Pozitia initiala a calului, adica i si j:');
     readln(ic,jc);
     write('Pozitia finala a calului, adica k si l:');
     readln(kc,lc);
     if (t[ic,jc]='*') or (t[kc,lc]='*') then begin
        writeln('Deplasare imposibila !'); halt
     end;
     for i:=1 to 8 do
         for j:=1 to 8 do a[i,j]:=65;
     a[ic,jc]:=0; pas:=0; modificare:=true;
     while modificare do begin
         modificare:=false;
         for i:=1 to 8 do
             for j:=1 to 8 do
                 if (a[i,j]=pas) then begin
                    if poate_sari(i,j,k,l,1,2)then marcheaza(k,l);
                    if poate_sari(i,j,k,l,-1,2)then marcheaza(k,l);
                    . . . . .
                 end;
         if a[kc,lc]<>65 then
            begin i:=8;j:=8;modificare:=false end; {daca am ajuns in
            situatia finala nu mai are rost sa calculam matricea}
         pas:=pas+1
     end;
     if a[kc,lc]=65 then writeln('Deplasare imposibila !')
     else begin
          writeln('Deplasarea din(',ic,',',jc,')in(',kc,',',lc,')');
          writeln('Se poate face prin ',a[kc,lc],' mutari.');
     end
end.
```


Scanned with OKEN Scanner

Problema 10.4 admite – aşa cum se vede – o rezolvare şi fără liniarizare de matrici. Există însă şi probleme în care acest lucru nu este posibil. Algoritmul folosit la rezolvarea problemei 10.5 se numeşte **algoritmul lui Lee** şi este o variantă mai „ciudată" a PD.

În acest algoritm se face de fapt o parcurgere a unui graf, cu dorinţa de a ne îndepărta cât mai puţin de punctul de plecare, şi în mod treptat. De obicei în algoritmul lui **Lee**, un nod reprezintă o stare a unui sistem, iar un arc – o transformare. Este o metodă de rezolvare a problemelor în care se cere numărul minim de operaţii necesare pentru a ajunge la un obiectiv.

În situaţia de faţă: matricea A[*i*, *j*] are semnificaţia că sunt necesare A[*i*, *j*] mutări ale calului pentru a ajunge din poziţia iniţială în poziţia (*i*, *j*). Elementul corespunzător poziţiei iniţiale are valoarea zero, iar pe noi ne interesează elementul care indică poziţia finală. Prin algoritmul lui Lee se calculează de fapt tot matricea drumurilor minime, dar într-un alt mod. Dezavantajul este acela că dacă dorim să refacem drumul soluţie, este necesară o structură auxiliară de date. În cazul nostru acest lucru nu este deloc supărător: mai luăm o matrice în care fiecare element atins arată de unde a fost atins.

Programul funcţionează în felul următor:

1. Se consideră că nu este nevoie de nici o mutare pentru a ajunge la starea iniţială, deci elementul corespunzător se consideră atins cu pasul 0;
2. Pentru toate elementele atinse la pasul anterior (aici 0) se calculează poziţiile în care se poate ajunge din ele; pentru fiecare astfel de poziţie, dacă nu a mai fost atinsă, se atinge acum la pasul curent (adică 1);
3. Problema se reia pentru pasul 1 ca pas anterior şi pasul 2 – pas curent etc;
4. Dacă la un pas *i* nu se mai poate ajunge la un nod nou, algoritmul se termină (în cazul problemei noastre, ieşirea din ciclul de marcare a nodurilor în paşi succesivi se face dacă în pasul curent nu s-a mai atins nici un nod nou).

Observaţie: Algoritmul lui Lee este mai performant decât cel de liniarizare, având ordinul N^2.

10.6 (VIII)

Se dă o pereche de numere de forma (*a*, *b*) asupra căreia putem face următoarele operaţii:

1. (*a*, *b*) —> (*a* – *b*, *b*)
2. (*a*, *b*) —> (*a* + *b*, *b*)
3. (*a*, *b*) —> (*b*, *a*).

Dându-se valorile (*a*, *b*) să se spună care este numărul minim de operaţii pe care trebuie să le facem astfel încât să ajungem la perechea (*c*, *d*), dacă acest lucru este posibil, sau „Nu se poate" dacă nu este posibil.

R 10.6 (Ordinul de complexitate al algoritmului: N^2)

```
const nmax=100;
      infinit=nmax*nmax+1;
var   t:array [1..nmax,1..nmax] of integer;
      pas,i,j,a,b,c,d:integer;  modificare:boolean;
```


Scanned with OKEN Scanner

```
function neatins(i,j:integer):boolean;
begin
  if (i in [1..nmax]) and (j in [1..nmax]) and (t[i,j]=infinit)
    then neatins:=true
    else neatins:=false
end;
procedure atinge(i,j:integer);
begin
  t[i,j]:=pas+1; modificare:=true
end;
begin
  write('Dati perechea a b ( ex. 4 6 )  : '); readln(a,b);
  write('Dati perechea c d ( ex. 12 8 )  : '); readln(c,d);
  for i:=1 to nmax do
    for j:=1 to nmax do t[i,j]:=infinit;
  t[a,b]:=0; pas:=0; modificare:=true;
  while modificare do begin
    modificare:=false;
    for i:=1 to nmax do for j:=1 to nmax do
      if t[i,j]=pas then begin
        if neatins(i+j,j) then atinge(i+j,j);
        if neatins(i-j,j) then atinge(i-j,j);
        if neatins(j,i) then atinge(j,i);
        if t[c,d]<>infinit then begin
           i:=nmax;j:=nmax;
           modificare:=false
        end
      end;
    pas:=pas+1
  end;
  if t[c,d]=infinit then
    writeln('Nu se poate ajunge la perechea data !')
  else  begin
    writeln('Este nevoie de ',t[c,d],' oper.pentru a ajunge de la');
    writeln('perechea (',a,',',b,') la perechea (',c,',',d,') !')
  end
end.
```

Se foloseşte strategia Lee, în care matricea T[*i*, *j*] are semnificaţia că plecând de la perechea iniţială, se poate ajunge la perechea (*i*, *j*) în T[*i*, *j*] moduri.

10.7 (IX)

Se dă o bară de lungime *n* ($1 \le n \le 40000$). Dorim să tăiem din ea bucăţi de dimensiuni prestabilite. Se dau *nr* astfel de dimensiuni în care putem să tăiem bara. Problema este în a tăia asemenea bucăţi din bară, de dimensiuni date, astfel încât să rămână cât mai puţin material care nu mai poate fi folosit (nu corespunde nici unei dimensiuni). Datele se citesc de la tastatură.

R 10.7 (Ordinul de complexitate al algoritmului: N)

```
type tablou=array [0..40000] of boolean;
     tablou_scurt=array [1..100] of word;
var a:tablou;
    u,l:tablou_scurt;
    i,j,n,nr:word;
```


Scanned with OKEN Scanner

```
procedure dinamic;
begin
  a[0]:=true;
  for i:=1 to n do a[i]:=false;
  for i:=0 to n do
    if a[i] then
      for j:=1 to nr do
        if i+l[j]<=n then a[i+l[j]]:=true
end;
begin
  write('Dati lungimea barei pe care o taiem :   '); readln(n);
  if n>nmax then begin
    writeln('Ati dat o dimensiune prea mare !');exit
  end;
  write('Cate tipuri de bare se pot taia :   '); readln(nr);
  writeln('Dati pe linia urmatoare lungimile celor ',nr,' repere:');
  for i:=1 to nr do read(l[i]); readln;
  dinamic;
  i:=n;
  while not a[i] do dec(i);
  writeln('Taierea optima se face cu pierdere de ',n-i,' unitati,');
  writeln('iar o solutie este: ');
  for j:=1 to nr do u[j]:=0;
  while i<>0 do begin
    j:=nr;
    while not ((l[j] <= i) and (a[i-l[j]]) ) do dec(j);
    inc(u[j]);
    i:=i-l[j]
  end;
  writeln;
  for i:=1 to nr do
    if u[i]>0 then writeln(u[i],' bucati de lungime ',l[i],'   ; ');
  writeln
end.
```

Metoda folosită este **Lee** într-o nouă variantă. Se construieşte un vector A cu *n* elemente, unde *n* este lungimea barei. În elementul A[*i*] reţinem True dacă o bară de lungime i se poate tăia exact în bucăţi de dimensiuni date. Iniţial A[0] = 0, şi pentru fiecare tip de bucată, A[0 + lungimea fiecărui tip de bucată[ ] = True. Dacă A[1] = True, adică s-a putut tăia o bară de lungime 1 prin tipurile de bucăţi date, atunci pentru fiecare tip de bucată pe care o putem tăia, A[1 + lungimea fiecărui tip de bucată] = True,... până când ajungem la o bară de lungime *n*. Căutăm apoi în vectorul A cea mai din dreapta poziţie pe care avem un element True, şi ştim cu precizie că cea mai bună soluţie pe care o putem obţine folosind tipurile de bucăţi date, este să tăiem o bară de lungime egală cu poziţia din vector care corespunde valorii True.

Dacă rezolvăm problema folosind metoda Greedy, observăm că în urma construirii vectorului A în mod recursiv, se pot afişa toate soluţiile posibile. Acesta este unul din dezavantajele PD: de obicei este găsită o singură soluţie, chiar dacă există mai multe. De obicei, combinând rezultatele PD cu recursivitatea putem găsi toate soluţiile optime.

Scanned with OKEN Scanner

10.8 (**IX**, Baraj, Bucureşti, 1995 – variantă)

Se dau două şiruri de maxim 100 de caractere fiecare. Problema cere să transformăm al doilea şir în primul folosind cât mai puţine operaţii, o operaţie fiind una din următoarele trei:

– se înlocuieşte o literă din al doilea şir cu una din primul şir;
– se inserează o literă în al doilea şir;
– se şterge o literă din al doilea şir.

R 10.8

```
program transformare;
const nmax=100;
var p,q:string[nmax]; i,j:integer;
    a:array [0..nmax,0..nmax] of integer;

procedure minim(i,j:integer);
var x,y,z,m:integer;
begin
  if (i=0) and (j=0) then exit;
  x:=a[i,j-1]; y:=a[i-1,j-1];
  z:=a[i-1,j]; m:=a[i,j];
  if m=x+1 then begin
    minim(i,j-1);
    writeln('inserare la :',j,' litera ',p[j])
  end
  else if m=z+1 then begin
         minim(i-1,j);
         writeln('stergere la :',j+1)
       end
       else begin
            minim(i-1,j-1);
            if m<>y then
               writeln('schimbare la :',j,' litera ',p[j])
       end
end;
function min(x,y,z:integer):integer;
var m:integer;
begin
  inc(x); inc(z);
  if q[i]<>p[j] then inc(y);
  if x<y then m:=x else m:=y;
  if z<m then m:=z;
  min:=m
end;
begin
  write('p: '); readln(p);
  write('q: '); readln(q);
  for i:=0 to length(p) do a[0,i]:=i;
  for i:=0 to length(q) do a[i,0]:=i;
  for i:=1 to length(q) do
    for j:=1 to length(p) do
      a[i,j]:=min(a[i,j-1],a[i-1,j-1],a[i-1,j]);
  minim(length(q),length(p))
end.
```


Scanned with OKEN Scanner

Se construieşte matricea A cu atâtea linii câte caractere are cel de-al doilea şir + 1, şi atâtea coloane câte caractere are primul şir + 1, cu semnificaţia A[*i*, *j*] = = numărul minim de operaţii ce trebuie făcute pentru a transforma primele *j* caractere din al doilea şir în primele *i* caractere din primul şir. Iniţial prima linie (linia 0) se umple cu valorile 0 1 ... nr_coloane, iar prima coloană se umple cu valorile 0 1... nr_linii.

Cele trei operaţii se reflectă în matricea A în felul următor:

– A[*i*, *j*] = A[*i* – 1, *j* – 1] + 1, dacă se înlocuieşte caracterul de pe poziţia *i* din al doilea şir cu caracterul de pe poziţia *j* din primul şir;

– A[*i*, *j*] = A[*i*, *j* – 1] dacă se inserează pe poziţia *i* în al doilea şir caracterul de pe poziţia *j* din primul şir;

– A[*i*, *j*] = A[*i* + 1, *j*] dacă se şterge caracterul de pe poziţia *i*.

În final elementul A[*nr_car_sir_2,nr_car_sir_1*] = numărul de operaţii care trebuie făcute; aceste operatii se pot reconstitui pe baza tabloului A.

10.9 (XII – C, Concurs, Timişoara, 1995)

O lege de compoziţie definită pe un alfabet $S = (c_1, c_2, c_3, ..., c_n)$ este dată într-un fişier text de intrare cu numele INPUT.DAT sub următoarea formă:

```
Cn
K11 K12 ... K1y
K21 K22 ... K2y
.................
Ky1 Ky2 ... Kyy
d
X
```

unde pe liniile 2, ..., *n* + 1 este dată tabla de compoziţie (cu spaţii între caractere), *d* este unul din elementele alfabetului, iar *X* este un şir cu elemente din *S*.

Problema constă în a realiza o parantezare a şirului *X* astfel încât rezultatul compunerii să fie *d*. Se face precizarea că legea nu este comutativă şi nici asociativă. În cazul în care există o astfel de parantezare, se va afişa „Se poate:", urmată de o soluţie. Dacă nu există soluţie, se afişează „Nu se poate!".

Exemple:

1. Pentru fişierul de intrare:

```
c
b c a
c b b
b b b
c
bbbba
```

o ieşire poate fi `Se poate: (b((bb)b))a`

2. Pentru fişierul de intrare:

```
c
b b a
b b b
b b b
a
bbbba
```

ieşirea este "Nu se poate !"

Scanned with OKEN Scanner

R 10.9

Problema este o generalizare a unei probleme propusă de Judeţul Braşov pentru finala de la Craiova (1994) şi dată la Alba Iulia la barajul de selecţie a lotului pentru Balcaniada din 1994 ([3]).

Observăm că având un şir de N elemente care trebuie parantezate, vor fi realizate $N - 1$ compuneri. Remarcăm de asemenea că datorită posibilităţii folosirii parantezelor, este respectat principiul optimalităţii în varianta a III-a (mixtă). De aceea încercăm să rezolvăm problema prin PD. Să începem prin a alege convenabil structurile de date.

O simplă matrice (bidimensională) nu poate rezolva problema. Noutatea constă în utilizarea unei matrici tridimensionale; drept urmare, metoda de rezolvare se va numi Programare dinamică 3D.

Se construieşte matricea A cu N linii, N coloane şi 26 nivele, cu următoarea semnificaţie: elementul A[i, j, c] (unde i este numărul liniei, j este numărul coloanei, c este un element al alfabetului) are valoarea True dacă între elementele i şi j ale şirului X se poate realiza o parantezare prin care să se ajungă la elementul c. Elementele matricii sunt calculate pe drepte paralele cu diagonala principală a bazelor şi în plane diferite.

Se calculează diagonala principală a planului cel mai de jos, apoi diagonala principală a planului de deasupra etc., apoi se începe cu dreapta de deasupra diagonalei principale şi paralelă cu aceasta etc., până când în final este calculată verticala A[1, N, *]. În acest moment putem afirma cu certitudine dacă se poate sau nu paranteza şirul dat, şi în caz afirmativ vom apela o procedură recursiva **divide et impera** care va afişa soluţia.

Această problemă este ceva mai abstractă, şi necesită o atenţie mărită.

```
const  nmax=30; {numarul maxim de elemente ale sirului X}
var    a:array[1..nmax,1..nmax,'a'..'z'] of boolean;
       fi:text; i,j,k,l,m,n:integer; X:string;
       t:array['a'..'z','a'..'z'] of char;
       last,dest,cc,c1,c2:char;

procedure cost;
begin
  for i:=1 to n do
    for j:=1 to n do
      for cc:='a' to last do
        a[i,j,cc]:=false;
  for i:=1 to n do a[i,i,x[i]]:=true;
  for k:=1 to n-1 do
    for i:=1 to n-k do begin
      j:=i+k;
      for l:=i to j-1 do
        for c1:='a' to last do
          for c2:='a' to last do
            if (a[i,l,c1]) and (a[l+1,j,c2]) then
                a[i,j,t[c1,c2]]:=true;
  end;
end;
```


Scanned with OKEN Scanner

```
procedure caut(s,d:integer; dest:char);
 var k:integer; c1,c2,c3,c4:char;
begin
  if s=d then begin
    write(x[s]); exit
  end;
  for k:=s to d-1 do
    for c1:='a' to last do
      for c2:='a' to last do
        if(a[s,k,c1])and(a[k+1,d,c2])and(t[c1,c2]=dest) then begin
          if k>s then write('(');
          caut(s,k,c1);
          if k>s then write(')');
          if k<d-1 then write('(');
          caut(k+1,d,c2);
          if k<d-1 then write(')');
          exit
        end
end;
begin
  assign(fi,'input.dat'); reset(fi);
  readln(fi,cc); last:=cc;
  for c1:='a' to cc do begin
    for c2:='a' to pred(cc) do read(fi,t[c1,c2],dest);
    read(fi,t[c1,cc]); readln(fi);
  end;
  readln(fi,dest); readln(fi,X);
  n:=length(x); cost;
  if a[1,n,dest] then begin
    writeln('Se poate: ');
    caut(1,n,dest); writeln;
  end else writeln('Nu se poate !');
  close(fi);
end.
```

10.10 (X – C, IOI, Suedia, 1994)

Se consideră un triunghi de numere în care pe prima linie apare un număr, pe a doua linie apar două numere, pe a treia linie se dau trei numere, etc. Scrieţi un program care să calculeze cea mai mare dintre sumele numerelor ce apar pe drumurile ce pleacă din vârf şi ajung la bază, astfel încât:

– în fiecare drum succesorul unui număr se află pe rândul de mai jos şi anume dedesubt sau pe diagonală la dreapta;

– numărul de rânduri este strict pozitiv, dar cel mult egal cu 100;

– toate numerele care apar în triunghi sunt întregi cuprinşi între 0 şi 99.

Exemplu: Pentru fişierul de intrare:

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

fişierul de ieşire trebuie să conţina valoarea 30.

Scanned with OKEN Scanner

R 10.10

```
uses crt;
const nmax=100;
type tab=array [1..nmax,1..nmax] of integer;
var t,a:tab;
    fi,fo:text;
    i,j,n:integer;
function min(a,b:integer):integer;
begin
  if a<b then min:=a else min:=b;
end;
procedure dinamic;
begin
  for i:=1 to n do t[n,i]:=a[n,i];
  for i:=n-1 downto 1 do begin
    for j:=1 to i do
      t[i,j]:=min(a[i,j]+t[i+1,j],a[i,j]+t[i+1,j+1]);
  end;
end;
begin
  assign(fi,'input.txt'); reset(fi);
  assign(fo,'output.txt'); rewrite(fo);
  readln(fi,n);
  for i:=1 to n do begin
    for j:=1 to i do read(fi,a[i,j]); readln(fi);
  end;
  dinamic;
  writeln(fo,t[1,1]);
  close(fi); close(fo);
end.
```

Vom aplica PD 2D folosind o matrice triunghi în care reţinem datele iniţiale şi – pe baza ci – vom construi o altă matrice triunghiulară T cu următoarea semnificaţie: T[*i*, *j*] = suma minimă pentru cel mai scurt drum care pleacă de pe o poziţie de pe ultima linie şi ajunge pe poziţia (*i*, *j*) în matricea iniţială. Relaţiile de recurenţă sunt:

```
T[i,j]=min(A[i,j]+T[i+1,j+1],A[i,j]+T[i+1,j]).
```

Metoda de PD folosită este varianta *înainte*. Ca o observaţie, încercaţi să scrieţi programul folosind metoda *înapoi*.

10.11 (**XI – C**, IOI, Argentina, 1993)

Se consideră o mulţime de oraşe situate în poziţii fixe pe harta ţării din care fac parte. Scopul este acela de a pleca din oraşul situat cel mai la vest, mergând numai spre est, până ajungem în oraşul situat cel mai la est, pentru ca apoi să ne întoarcem în oraşul din care am plecat.

Orice oraş este vizitat cel mult o dată (exceptând oraşul iniţial, care apare de două ori: la plecare şi la sosire). Programul trebuie să găsească un asemenea circuit astfel încât numărul de oraşe vizitate să fie maxim.

Datele de intrare sunt date în fişierul INPUT.TXT în forma:

Scanned with OKEN Scanner

```
n v
i j
k l
...
z w
```

unde n este numărul de oraşe, v este numărul de legături directe de tipul oraş i-oraş j, (numărul de muchii ale grafului asociat), după care urmează v linii, conţinând fiecare câte două numere, corespunzând unei perechi de două oraşe între care există drum direct.

Ieşirea: Pe ecran se va afişa lungimea celui mai lung circuit, sau mesajul `Nu exista circuit!`.

Se consideră că oraşele sunt codificate de la vest la est în ordine crescătoare şi nu există două oraşe situate pe acelaşi meridian.

R10.11.

```
uses crt;
const nmax=100;
type tab=array [1..nmax,1..nmax] of integer;
     tabb=array [1..nmax,1..nmax] of boolean;
var a:tab; g:tabb; fi:text;
    i,j,k,n,v:integer;
procedure cost;
begin
  for i:=1 to n do
      for j:=1 to n do a[i,j]:=0;
  a[1,1]:=1;
  for i:=1 to n-1 do
    for j:=i+1 to n do
      for k:=1 to j-1 do
        if g[j,k]and(a[i,k]>0)and(a[i,j]<a[i,k]+1) then begin
          a[i,j]:=a[i,k] + 1; a[j,i]:=a[i,j];
        end
end;
begin
 assign(fi,'input.txt'); reset(fi);
 readln(fi,n,v);
 for i:=1 to n do
     for j:=1 to n do g[i,j]:=false;
 for k:=1 to v do begin
   readln(fi,i,j);
   g[i,j]:=true;g[j,i]:=true
 end;
 cost; k:=0;
 for i:=1 to n-1 do
   if g[i,n] and (k < a[i,n]) and (a[i,n] > 0) then k:=a[i,n];
 if k>0 then writeln('Lung. celui mai lung circuit este ',k,' !')
        else writeln('Nu exista circuit !');
 close(fi)
end.
```

Metoda de PD folosită este varianta „înapoi”. Se cere de fapt găsirea a două drumuri independente, unul care duce de la vest la est, celălalt de la est la vest, astfel încât numărul de oraşe vizitate să fie maxim şi să se treacă prin oraşul n.

Scanned with OKEN Scanner

Construim matricea A[*i*, *j*] cu semnificaţia A[*i*, *j*] = numărul de oraşe ale unui circuit care porneşte din oraşul 1 (cel mai la vest) şi trece prin oraşele *i* şi *j*. Evident, A[*i*, *j*] = A[*j*, *i*].

Formula de recurenţă este:

```
A[i,j]=max(A[i,k]+1)
```

unde (*k*, *j*) este un drum direct între două oraşe (o muchie în graful G) şi A[*i*, *k*] > 0 (adică adăugăm un nou oraş la un circuit care pleacă din oraşul 1, trece prin oraşele *i* şi *k* şi avem un drum direct între oraşele *k* şi *j*). În acest moment există deci un circuit care pleacă din 1, trece prin *i*, prin *k* şi înapoi prin *j*.

Numărul căutat de noi, adică lungimea celui mai lung circuit care pleacă din 1 şi trece şi prin *n*, se află în matrice pe ultima coloană, fiind maximul acestei coloane. Ordinul algoritmului este N^3.

10.12 (**XII**,Baraj IOI, Bucureşti, 1995)

În practică apare în cazul editoarelor de texte necesitatea de a realiza operaţii complexe asupra unor secvenţe de text. Una din aceste funcţii este aceea de a aranja textul în pagină, astfel încât spaţiile să fie cât mai optim puse.

Trebuie să realizăm un program care face următoarele:

– primeşte *n* cuvinte de la tastatură;

– citeşte lăţimea *p* a paginii pe care va aşeza cele *n* cuvinte;

– aşează cele *n* cuvinte în pagină astfel încât suma cuburilor numărului de spaţii de la sfârşitul fiecărui rând (sau 0 dacă nu mai rămân spaţii la sfârşitul rândului), mai puţin ultimul rând, să fie minimă.

A fost aleasă funcţia $F(X) = X^3$ deoarece reflectă mai bine cât de bine sunt aşezate cuvintele în pagină, astfel încât în ansamblu să rămână cât mai puţine spaţii la sfârşitul rândurilor.

R 10.12 (Ordinul de complexitate al algoritmului este N)

```
uses crt;
const nmax=100; { numarul maxim de cuvinte }
type cuvant=string[100];
     tab=array [1..nmax+1] of longint;
var  p,i,j,k,l,n:longint;
     c:array [1..nmax] of cuvant;
     a,b:tab;
function suma(i,j:integer):integer;
 var k,l:integer;
begin
 l:=0;
 for k:=i to j do l:=l+length(c[k]) + 1;
 l:=l-1; suma:=l;
end;
procedure dinamic;
 var i,j,k,l:longint;
begin
 a[n+1]:=0;
```


Scanned with OKEN Scanner

```
 for i:=n downto 1 do begin
   a[i]:=maxlongint;
   for j:=i to n do
     if suma(i,j)<=p then
     if j=n then begin a[i]:=0; b[i]:=n; end
     else begin
        k:=suma(i,j);
        l:=(p-k)*(p-k)*(p-k) + a[j+1];
        if a[i] > l then begin
          a[i]:=l;
          b[i]:=j;
        end
     end
 end
end;
begin
 write(#13#10,'Numarul de cuvinte: '); readln(n);
 writeln('Dati cele ',n,' cuvinte, fiecare pe cate o linie:');
 for i:=1 to n do readln(c[i]);
 write('Latimea paginii pe care vor fi puse cuvintele: ');readln(p);
 dinamic; writeln('Suma optima: ',a[1]); i:=1;
 while i<=n do begin
   for j:=i to b[i] do write(c[j],' '); writeln;
   i:=b[i]+1;
 end;
 writeln
end.
```

Este aplicată metoda PD în varianta *înainte*. Se construieşte şirul A[*i*] cu *n* elemente, având semnificaţia A[*i*] = suma minimă a cuburilor numărului de spaţii de la sfârşitul tuturor rândurilor, mai puţin ultimul, dacă se folosesc doar cuvintele *i*, *i* + 1, *i* + 2, ..., *n* şi dacă cuvântul *i* se află primul pe rândul curent.

Relaţia de recurenţă este:

```
A[i]=min(A[j+1]+SUMA(i,j))
```

unde *j* este ultimul cuvânt de pe linia pe care se află cuvântul *i*, iar SUMA(*i*, *j*) este o funcţie care arată cât este de lungă linia formată din cuvintele *i*, *i* + 1, *i* + 2, ..., *j*, împreună cu spaţiile aferente (între două cuvinte se află un spaţiu).

Observaţie: Pentru ca programul să fie utilizabil practic, se poate modifica ieşirea astfel încât spaţiile să nu fie afişate la sfârşitul liniei, ci să fie introduse între cuvinte în mod proporţional. Lăsăm ca exerciţiu acest fapt.

10.13 (X – C, Olimpiadă,Bucureşti, 1995)

Pe intervalul [1,*a*] cu *a* număr natural vom numi diviziune întreagă aditivă sistemul de numere întregi (A_0 A_1 .. A_n) cu $1 = A_0 < A_1 < ... < A_n = a$ şi pentru orice *k* există *i* şi *j* cu $0 \le i \le j < k \le n$ astfel încât $A_k = A_i + A_j$. Să se elaboreze un program care pentru un număr natural *a* ($a > 1$) dat de la tastatură, determină diviziunea aditivă de lungime minimă pentru intervalul [1, *a*].

R 10.14 (Soluţie dată de Gheorghioiu Ovidiu, Alba Iulia).

```
const nmax=255;
type mult=set of 1..nmax;
```


Scanned with OKEN Scanner

```
var y:array[1..nmax] of mult;
    i,j,a,n,m,mi,mj:integer;
function card(a:mult):integer; {determina cardinalul multimii "a" }
var i,nn:integer;
begin
     nn:=0;
     for i:=1 to n do if i in a then inc(nn);
     card:=nn
end;
begin
     write('n=');readln(n);
     y[1]:=[1];
     for i:=2 to n do begin
         m:=maxint;
         for j:=i div 2 downto 1 do begin
             mi:=card(y[j]+y[i-j]);
             if mi<m then begin
                m:=mi; mj:=j
             end;
         end;
         y[i]:=y[mj]+y[i-mj]+[i]
     end;
     writeln('Costul minim: ',card(y[n])-1);
     for i:=1 to n do if i in y[n] then write(i,' ');
     readln
end.
```

Problema a fost dată inițial la prima ediție a Balcaniadei de Informatică (Constanța 1994), fiind propusă de delegația Bulgariei. Pentru rezolvare s-a folosit PD în varianta *înapoi*. Construim șirul de mulțimi Y, unde Y[*i*] = cea mai scurtă diviziune aditivă pentru intervalul [1, *i*], Y[1] = [1]. Relația de recurență este:

$$Y[i]=\min(Y[j]+Y[i-j]+[i]) \text{ cu } 1 \le j \le [i/2],$$

relație care înseamnă: pentru a face o diviziune aditivă de lungime minimă pentru intervalul [1, *i*] se încearcă toate posibilitățiile de a alătura două diviziuni: una pentru intervalul [1, *j*], alta pentru intervalul [1, $i - j$], iar prin combinarea celor două se ajunge la o diviziune aditivă pentru intervalul [1, *i*] (deoarece $j + i - j = i$). Dintre toate combinațiile de câte două diviziuni, o vom alege pe cea care are o lungime minimă.

Probleme propuse

10.14 (X – C, Concursul Tuyaamada, Irkuțk 1995)

O galerie de pictură are nxm camere egale. Fiecare din ele are uși de comunicare cu camerele vecine. La intrarea în fiecare cameră un vizitator plătește o anumită sumă de bani, în funcție de valoarea picturilor din cameră. Intrarea în galerie se face numai pe ușa (1,1) iar ieșirea este unică: prin camera (*n*, *m*).

Să se scrie un program care să determine cel mai scump traseu.

Cerințe: *n* și *m* se dau la tastatură. Prețurile pentru vizitarea camerelor sunt numere întregi pozitive de maxim două cifre, care se obțin folosind o rutină de

Scanned with OKEN Scanner

generare aleatoare a numerelor. Drumul vizitatorului prin galerie se dă ca o succesiune de perechi ale coordonatelor camerelor.

Exemplu:

```
Intrare:   n=4, m=5
Iesire (pe ecran):
                         1     2     3     4     5
                       -------------------------------
            ->  1  |    20    10    32    17    19
                2  |     9    23    25    26    13
                3  |    21    14    19    30    34
                4  |    12    41     9    26    56   ->
Traseu:
(1,1) -> 20
(1,2) -> 30
(1,3) -> 62
(2,3) -> 87
(2,4) -> 113
(3,4) -> 143
(3,5) -> 177
(4,5) -> 233
```

10.15 (IX, [2])

Se consideră o tablă de şah de dimensiune $N \times N$ ($N \leq 100$) pe care sunt dispuse obstacole. Se cere să se tipărească numărul minim de mutări necesare unui nebun pentru a se deplasa, respectând regulile jocului de şah şi ocolind obstacolele, dintr-o poziţie iniţială dată într-o poziţie finală dată.

Indicaţie: vezi problema calului (10.5).

10.16 (XI)

O funcţie f : ℕ -> ℕ este strict crescătoare şi are proprietatea că, pentru orice $k \leq 1$, cel de-al k-lea întreg strict pozitiv care nu apare în şirul f(1), f(2), f(3), f(4), ... este f(f(k)) + 1. Determinaţi funcţia f.

Exemplu: f(1)=1, f(2)=3, f(5)=8.

Indicaţie: Se respectă definiţia funcţiei f, pentru care se construieşte şirul f[1], f[2],.. Metoda de PD este varianta „înapoi". Funcţia f poate fi calculată recurent, având formula: $f(x) = x\frac{1+\sqrt{5}}{2}$.

10.17 (X – C, Olimpiada Europei Centrale şi de Est, Ungaria, 1995)

Vasele 1, 2, 3 au volumele A1, A2, A3. Iniţial vasul 1 este plin cu apă. Se poate face o singură operaţie şi anume se toarnă apa dintr-un vas până când acesta se goleşte sau până când vasul în care se toarnă se umple.

Problema este ca dintr-un număr minim de asemenea operaţii să se ajungă la situaţia ca într-unul din vase să se afle o cantitate Q de apă.

Indicaţie: Se rezolvă cu algoritmul lui prin Lee, A[i, j, k] =numărul minim de operaţii pentru a ajunge din situaţia iniţială în situaţia unde în primul vas se află i unităţi de apă, în al doilea vas j unităţi de apă, iar în al treilea k unităţi de apă. Algoritmul se încheie când A[Q, i, j], A[i, Q, j] sau A[i, j, Q] devine iniţializat.

10.18 (**XI**, Olimpiadă, Iugoslavia, 1994)

Fie funcţiile f, g definite pentru numere întregi astfel: f(a) = [a/2] (parte întreagă din a împărţit la 2), g(a)=3 • a. Să se determine din câte aplicări ale funcţiilor f şi g se poate ajunge de la numărul X la numărul Y, ambele date iniţial.

Indicaţie: Se construieşte şirul S[i] = numărul minim de operaţii necesare pentru a ajunge de la X la numărul i. Iniţial S[X]=0 iar toate celelalte elemente ale şirului se consideră neiniţializate.

Algoritmul se încheie în momentul în care se iniţializează S[Y] sau când prin operaţiile de extindere nu se mai pot iniţializa alte elemente ale şirului S.

10.19 (**X – C**, Barajul Balcaniadă, 1994, Alba Iulia)

Se introduce de la tastatură un număr natural N ≤ 100. Se cere să se calculeze numărul şirurilor de caractere de lungime N care conţin doar paranteze rotunde deschise şi închise, cu proprietatea că parantezele se potrivesc corect.

Exemplu: pentru N = 4 se afişează 2.

( ) ( )

(( ))

Indicaţie: Se foloseşte şirul S, cu $S(0) = 1$ şi relaţia de recurenţă:

$$S(i) = \sum_{0 \le j \le i \le N} S(j) \cdot S(i-j).$$ În $S(N+1)$ se obţine numărul dorit.

10.20 (**XII – C**, Baraj IOI, Craiova, 1994)

Pe produsul cartezian $\mathbb{N} \times \mathbb{N}$ se defineşte operaţia: $(a, b)(b, d) = (a, c)$ $\forall a, b, c \in \mathbb{N}$. Această lege este asociativă: [(a, b) (b, c)] (c, d) = (a, b) [(b, c) (c, d)], iar efectuarea ei necesită a * b * c secunde.

Fiind dat un şir $X_1, X_2, X_3, ..., X_n$, $n \ge 3$, se cere timpul minim şi cel maxim în care se poate efectua produsul, folosind asociativitatea legii.

Exemplu: Pentru intrarea

```
7  1  9  3
```

răspunsul este:

```
Timp minim: 48 secunde
Timp maxim: 4 minute si 12 secunde
```

Indicaţie: Se construieşte matricea A[i,j] cu semnificaţia: numărul minim (maxim, depinde care problemă se rezolvă) de secunde pentru a efectua produsul elementelor i, i+1, i+2, ..., j.

```
A[i,i]=0, A[i,i+1]=0,
A[i,j]=min(A[i,k]+A[k,j]+X[i]*X[k]*X[j]),i≤k≤j.
```

În A[1,N] se obţine rezultatul dorit. Ca o observaţie, se aplică PD în varianta mixtă, iar problema se aseamănă foarte mult cu problema *produs optim de matrici*, care este deja celebră.

10.21 (**XI – C**, Baraj IOI, Craiova 1994)

Spunem că x este supus lui y dacă reprezentarea în baza 2 a lui x se obţine din reprezentarea în baza 2 a lui y numai prin ştergerea unor biţi. Problema constă în a determina, pentru x şi y daţi, cel mai mare număr care este supus şi lui x şi lui y.

Indicaţie: Se rezolvă PD dacă se adaptează algoritmul care găseşte cel mai lung subşir comun a, în sensul că pe poziţia A[i, j] nu mai reţinem numărul de elemente ale celui mai lung subşir comun

Scanned with OKEN Scanner

format din primele i elemente din primul şir şi primele j elemente din al doilea şir, ci se păstrează cel mai mare număr care se obţine din reprezentarea în baza 2 a primelor i poziţii din primul şir şi a primelor j poziţii din al doilea şir, adică se reţine supusul.

10.22 (XI – C, Olimpiada de Informatică, Slatina, 1995)

O regiune deşertică este reprezentată printr-un tablou de dimensiune $m \times n$, $m \leq 100$, $n \leq 100$. Elementele tabloului sunt numere naturale mai mici decât 255 reprezentând diferenţele de nivel faţă de nivelul mării (cota zero).

Să se stabilească:

(a) Un traseu pentru a traversa deşertul de la Nord la Sud (de la linia 1 la linia m) astfel:

- se porneşte dintr-un punct al liniei 1;
- deplasarea se poate face în una din direcţiile E, S–E, S, S–V, V;
- suma diferenţelor de nivel (la urcare şi la coborâre) să fie minimă.

(b) Un traseu pentru a traversa deşertul de la Nord la Sud în condiţiile punctului (a), la care adăugăm condiţia:

- lungimea traseului să fie minimă.

Fişierul de intrare DESERT.TXT conţine mai multe seturi de date separate prin caracterul *, fiecare având următoarea structură:

```
linia 1:      m,n
linia 2:
...           elementele tabloului (pe linii) separate prin spatii
linia m:
```

Fişierul de ieşire TRASEU.TXT va conţine pentru fiecare set de date:

(a)

TRASEU: (i_1, j_1) (i_2, j_2) ... (i_p, j_p)

(b)

TRASEU: (i_1, j_1) (i_2, j_2) ... (i_k, j_k)

Exemplu:

```
Fisierul DESERT.TXT
4  4
1  4  4  3
5 50 75  4
6  5  4  4
6  5  1 10
*
Fisierul TRASEU.TXT
(a)
TRASEU: (1,3) (2,4) (3,4) (3,3) (4,2)
(b)
TRASEU: (1,2) (2,1) (3,2) (4,2)
```


Scanned with OKEN Scanner

XI. ALOCARE DINAMICĂ

Probleme propuse

11.1 (VIII)

Se pun în memorie (heap) două numere. Să se interschimbe valorile lor utilizând numai adrese.

R 11.1

Tipul anonim ^real desemnează un tip referinţă (adresă) a unui număr real. Deci a, b, c reprezintă nişte repere de memorie unde pot fi cazate numere reale (câte 6 octeţi). Notaţia a^ desemnează conţinutul celor 6 octeţi referiţi de a (deci valoarea reală găzduită de variabila a). Nu trebuie interpretată greşit semnificaţia instrucţiunilor de citire/scriere ca şi cum citirea sau scrierea variabilelor desemnate prin adrese s-ar face altfel decât a celorlalte variabile: readln(a^) reprezintă tot citirea de la tastatură a unei valori numerice care va fi depusă în heap la adresa a.

```
program interschimba;
var a,b,c:^real;
begin
     new(a);
     new(b);
     write('Valoarea primului numar:');
     readln(a^);
     write('Valoarea celuilalt numar:');
     readln(b^);
     new(c);
     c^:=a^;  {la adresa c se copiază valoarea de la adresa a}
     a^:=b^;
     b^:=c^;
     writeln (a^,b^)
end.
```

11.2 (X)

Se consideră programul:

```
program adrese;
var a,b:real;
    c,d:^real;
begin
     write('Valoarea primului numar:');
     readln(a);
     write('Valoarea celuilalt numar:');
     readln(b);
     c:=@a;
     d:=@b;
     a:=d^;
     b:=c^;
     writeln (a,b)
end.
```

Care va fi efectul execuţiei lui?

Scanned with OKEN Scanner

R 11.2

Simplul fapt că operăm interschimbarea folosind adresele variabilelor statice a şi b nu rezolvă problema celui de-al treilea pahar (din prezentarea anecdotică a metodei de interschimbare a valorilor a două variabile folosind o a treia de manevră). Efectul va fi că şi a şi b au aceeaşi valoare la sfârşitul execuţiei, lucru care va apărea şi pe ecran. Deci, dacă dorim cu orice preţ să lucrăm cu adrese, trebuie să „inventăm" o adresă nouă (şi aici ne-am permis fantezia să o creem ca referinţă de variabilă dinamică):

```
new(c);
c^:=a;
d:=@b;
a:=d^;
b:=c^
```

11.3 (X)

Să se genereze o listă cu un număr dat de componente numere reale oarecare şi să se determine prin parcurgerea listei câte numere pozitive sunt.

R 11.3

O prelucrare similară se realizează cu un tablou alocat static, de exemplu `v:array[1..10000]of real`, însă consumul de memorie prin alocarea inutilă a unui spaţiu de 6*10000 de octeţi (când vom utiliza poate numai câteva dintre cele 10000 de poziţii anunţate) precum şi limitarea spaţiului variabilelor la cei 64K (segmentul de date) fac ca alocarea statică să fie de foarte multe ori o metodă „fără viitor" pentru multe dintre probleme (în special cele care solicită atât spaţiu de memorie vast – pentru care există alternativa lucrului cu fişiere – cât şi manevre de proporţii în memorie).

Vom construi lista alocând *n* spaţii de câte 10 octeţi (6 – numărul real şi 4 – adresa elementului următor) cu grija de a memora separat adresa primului element al listei (prima) şi de a asigura continuitatea listei prin atribuirea corectă a adresei următoare pentru fiecare element p^ al listei. Lipsa preocupării de a atribui *nil* câmpului de legătură pentru ultimul element al listei (marcând astfel sfârşitul listei) se justifică numai pentru acele probleme în care cunoaşterea exactă a numărului de elemente ale listei în fiecare moment face ca testarea sfârşitului listei să fie inutilă. În general însă o listă suferă prelucrări care fac variabilă lungimea ei şi, dacă nu se actualizează tot timpul o valoare care să memoreze numărul de componente ale listei, va deveni imperios necesară atribuirea virtuală `a^.next:=nil` după fiecare atribuire `a^.nr:=oricare`.

```
program liste;
type adresa=^lista;
     lista=record
                nr:real;
                next:adresa
           end;
var a,b,prima:adresa;
    n,i,ap:integer;
```


Scanned with OKEN Scanner

```
function oricare:real;
var o:real;
begin
   o:=100*random;
   if random(2)=0 then oricare:=-o
                  else oricare:=o
end;
begin
   write('INTRODUCETI NR DE ELEMENTE:');readln(n);
   randomize;
   new(a);
   prima:=a;
   a^.nr:=oricare;
   write(a^.nr:8:2);
   for i:=2 to n do begin
                       b:=a;
                       new(a);
                       a^.nr:=oricare;
                       write(a^.nr:8:2);
                       b^.next:=a
                    end;
   writeln;
   ap:=0;b:=prima;
   for i:=1 to n do begin
                       if b^.nr>0 then ap:=ap+1;
                       b:=b^.next
                    end;
   writeln('SUNT ',ap,' NUMERE POZITIVE')
end.
```

11.4 (X)

Să se creeze o listă conţinând numele şi mediile a *n* elevi, listă ce va fi completată prin adăugări succesive de noi elemente pe poziţii date. La fiecare pas se va prezenta configuraţia listei validând eventualele cereri incorecte de adăugare sau eventualele valori necorespunzătoare ale mediilor.

R 11.4

Printr-o succesiune logică de apeluri ale procedurilor de creare, listare şi adăugare cu restricţiile corespunzătoare asupra valorilor introduse problema este rezolvată. Sugerăm utilizarea unei astfel de prelucrări în contextul unui ansamblu de alte prelucrări pentru a o face cu adevărat utilă (de exemplu datele să se citească dintr-un fişier şi la sfârşit să se salveze într-un fişier).

```
program lista;
type adresa=^elev;
     elev =record
                  nume:string[30];
                  media:real;
                  next:adresa
     end;
var prima,p,q,r:adresa;
    n,i,s:integer;
```


Scanned with OKEN Scanner

```
procedure citeste(a:adresa);
begin
     write('Numele:');readln(a^.nume);
     repeat
          write('Media:');readln(a^.media)
     until (a^.media>=1)and(a^.media<=10)
end;
procedure creare;
var i:integer;
begin
     new(prima); p:=prima;
     citeste(p);
     p^.next:=nil;
     for i:=2 to n do
         begin
              new(q);
              p^.next:=q;
              citeste(q);
              q^.next:=nil;
              p:=q;
         end
end;
procedure adauga_dupa(k:integer);
var i:integer;
begin
     p:=prima;
     for i:=2 to k do p:=p^.next;
     new(q);
     citeste(q);
     r:=p^.next;
     p^.next:=q;
     q^.next:=r
end;
procedure afisare;
begin
   p:=prima; i:=1;
   while p<>nil do
      begin
        writeln(i:3,'.',p^.nume,'':30-length(p^.nume),p^.media:6:2);
        p:=p^.next;i:=i+1
      end
end;
begin
     writeln('Cati elevi sunt?');readln(n);
     creare; afisare;
     repeat
          write ('Dupa al catelea elev adaugati? (0-->iesire)');
          readln(s);
          if s in [1..i-1] then adauga_dupa(s);
          afisare
     until s=0
end.
```

Observaţie: Atragem atenţia asupra importanţei declarării variabilelor *i* şi *n* ca variabile globale (utilizările lui *i* în proceduri nesuprapunându-se) şi a inutilităţii parametrului *k* în cazul în care se rescrie întreaga procedură înlocuind pe *k* cu *s*.

Scanned with OKEN Scanner

11.5 (X)

Să se genereze aleator o listă cu un număr dat de elemente (caractere alfabetice mari) şi să se afişeze elementele listei în ordinea în care au fost generate şi în ordine inversă generării.

R 11.5

Se creează o listă de tip stivă în care elementele se introduc de la un capăt al listei şi se extrag prin acelaşi capăt. La generare şi introducere în listă se afişează valorile (în ordine), apoi prin parcurgerea stivei se obţin valorile în ordine inversă. Dacă odată cu parcurgerea stivei dorim să o şi eliberăm extrăgând propriu-zis fiecare element din vârful stivei, vom utiliza procedura *dispose* care eliberează spaţiul de memorie ocupat de o variabilă dinamică (instrucţiunile corespunzătoare nefiind utile în contextul problemei noastre le-am scris sub formă de comentarii).

```
program caractere;
type adresa=^caracter;
     caracter=record
                    car:char;
                    urm:adresa
              end;
var prima,p,q:adresa;
    n,i:longint;
begin
     write('Cate elemente are lista ?');readln(n);
     prima:=nil;
     for i:=1 to n do begin
         new(p);
         p^.car:=chr(65+random(26));write(p^.car);
         p^.urm:=prima;prima:=p
     end;
     writeln;
     while prima<>nil do begin
           write(prima^.car);{p:=prima;}
           prima:=prima^.urm;{dispose(p)}
     end; writeln
end.
```

11.6 (X)

Se citesc elementele a două şiruri de numere reale. Sfârşitul fiecărui şir este marcat prin prezenţa pe poziţia *i* a numărului *i*. Să se formeze din toate elementele celor două şiruri o listă dublu înlănţuită care să aibă elementele ordonate şi să se afişeze atât în ordine crescătoare cât şi în ordine descrescătoare.

R 11.6

Deoarece nu avem în perspectivă nici o altă prelucrare cu şirurile date, nu le vom organiza de sine stătător ci le vom prelucra pe măsură ce ne sunt furnizate elementele lor. Vom iniţializa o listă dublu înlănţuită vidă în care vom introduce prin inserţie componentele celor două şiruri. Pentru a fi mai uşor de urmărit, am realizat inserarea distinct în fiecare dintre cazurile limită: în listă vidă şi adăugare la unul dintre capete.

Scanned with OKEN Scanner

```
program ordonare_ldi;
type adresa=^numar;
     numar=record
                  nr:real;
                  pre,urm:adresa
           end;
var cap1,cap2,p,q:adresa; x:real;
procedure inserare;
begin
     if cap1=nil then begin
        cap1:=q;           {---------------------------}
        cap2:=q;exit       {inserare la inceput de lista}
     end;                  {---------------------------}
     p:=cap1;
     if p^.nr>q^.nr then begin
        q^.urm:=cap1;      {---------------------------}
        cap1^.pre:=q;      {inserare la inceputul listei}
        cap1:=q; exit      {---------------------------}
     end;
     while (p<>nil)and(p^.nr<q^.nr) do p:=p^.urm;
     if p=nil then begin
        q^.pre:=cap2;      {---------------------------}
        cap2^.urm:=q;      {inserare la sfarsitul listei}
        cap2:=q; exit      {---------------------------}
     end;
     q^.pre:=p^.pre;q^.urm:=p;  {----inserare in lista---}
     p^.pre^.urm:=q;p^.pre:=q
end;
procedure prelucrare(c:char);
var i:word;x:real;
begin
     i:=1;
     write('Elementul ',i,' al sirului ',c,':'); readln(x);
     while x<>i do begin
           new(q); q^.pre:=nil; q^.urm:=nil;
           q^.nr:=x; inserare; i:=i+1;
           write('Elementul ',i,' al sirului ',c,':'); read(x)
     end
end;
procedure listare_cresc;
begin
     p:=cap1;
     while p<>nil do begin
           write(p^.nr:10:2);
           p:=p^.urm
     end; writeln
end;
procedure listare_descresc;
begin
     p:=cap2;
     while p<>nil do begin
           write(p^.nr:10:2);
           p:=p^.pre
     end; writeln
end;
```


Scanned with OKEN Scanner

```
begin
     writeln(' Se marcheaza sf.fiecarui sir furnizand o valoare');
     writeln('  egala cu indicele la care s-a ajuns (a[4]=4)');
     cap1:=nil;cap2:=nil;
     prelucrare('A');
     prelucrare('B');
     listare_cresc;
     listare_descresc
end.
```

11.7 (X)

Se citesc mai multe numere cu care se formează o listă circulară. Sfârşitul listei este marcat prin furnizarea de la tastatură a unui element care este suma precedentelor două numere.

a) Să se afişeze până la apăsarea unei taste elementele listei cu pauză de 1/2 secundă între ele. Dacă nu se apasă nici o tastă, ele vor fi afişate la nesfârşit.

b) Aceeaşi cerinţă dar se cere ca afişarea să se facă în ordinea inversă introducerii.

Exemplu: Se citesc: `6 -2 5.5 25 1 26` (26 = 1 + 25 --> stop)
şi se afişează: `6 -2 5.5 25 1 26 6 -2 5.5 25 1 ... (a)`
şi respectiv: `26 1 25 5.5 -2 6 26 1 25 5.5 -2 ...(b).`

R 11.7

Se formează iniţial o listă de două elemente care se succed reciproc şi apoi se citeşte câte un element inserându-se după elementul curent:

```
program circulara;
uses crt;
type adresa=^element;
     element=record
                  nr:real;
                  urm:adresa
             end;
var p,q:adresa;
    vstop:real;
procedure citeste(a:adresa);
begin
     write('Dati un nr.real : ');
     readln(a^.nr)
end;
begin
     new(p);citeste(p);
     new(q);citeste(q);
     p^.urm:=q;
     q^.urm:=p;
     repeat
          vstop:=p^.nr+q^.nr;
          writeln('Pentru iesire dati valoarea: ',vstop:10:2);
          p:=q; new(q); citeste(q);
          q^.urm:=p^.urm;
          p^.urm:=q
     until q^.nr=vstop;
```


Scanned with OKEN Scanner

```
    repeat
        q:=q^.urm;write(q^.nr:10:2);delay(500)
    until keypressed;
    writeln
end.
```

Observaţie: Pentru punctul (b), deşi aparent se schimbă doar locul de inserare şi eventual din legătură_la_succesor (câmpul urm) este suficient să facem legătură_la_predecesor, lucrurile nu mai sunt atât de „simetrice". Se mai declară o adresă de element *r* care va asigura închiderea înapoi a listei când se inserează un nou element.

```
begin
    new(p);citeste(p);
    new(q);citeste(q);
    p^.urm:=q;
    q^.urm:=p;
    r:=p;
    repeat
        vstop:=p^.nr+q^.nr;
        writeln('Pentru iesire dati valoarea: ',vstop:10:2);
        p:=q;                    {*}
        new(q);citeste(q);       {*}
        q^.urm:=p;               {*}
        r^.urm:=q                {*}
    until q^.nr=vstop;
    repeat
        write(q^.nr:10:2);
        delay(500);
        q:=q^.urm
    until keypressed;
```

10.8 (X)

Un ţăran gospodar are un sistem propriu de a-şi organiza fondurile băneşti ale gospodăriei: el îşi depune sumele încasate pentru produsele vândute în diferite zile în plicuri distincte scriind pe fiecare plic data când a obţinut suma respectivă. Când are de efectuat cheltuieli pentru mica sa "fermă", el ia suma necesară din ultimele depuneri desfiinţând plicurile care nu mai conţin nici un leu şi modificând data plicurilor pe care le-a descompletat (în locul datei depunerii el consemnează pe plic data în care a cheltuit o parte din banii existenţi în plicul respectiv). Ţăranul nostru are trei principii: ultimii bani încasaţi sunt cei mai buni de investit sau de cheltuit, să nu cheltuiască mai mult decât are (prin sistemul de împrumuturi) şi acela că nu este spornic să iei bani din două locuri deodată (deci nu va umbla niciodată la alt plic decât dacă nu are suficienţi bani în ultimul). Să se înregistreze depunerile şi cheltuielile ţăranului afişând în final plicurile existente şi data înscrisă pe fiecare.

R 10.8

Sistemul de lucru al ţăranului ne duce cu gândul la o stivă în care pune plicuri cu bani (procedura push) şi din care ia bani (procedura pop). Pentru a nu încerca extragerea unei sume mai mari decât totalul disponibil, variabila *s*. actualizată la fiecare operaţie pe stivă, reprezintă la orice moment suma totală existentă în plicuri:

Scanned with OKEN Scanner

```
program economii;
uses crt;
type adresa=^plic;
     plic=record
            suma:real;
            data:record zi:1..31; lu:1..12; an:1900..2020  end;
            next:adresa
     end;
var p,q,vs:adresa;          {vs=adresa reprezentând vârful stivei}
    d:plic;s:real;c:char;
procedure consult;
var p:adresa;
begin
     clrscr;p:=vs;
     writeln('Suma totala disponibila:',s:14:2);
     while p<>nil do
           with p^,data do begin
                writeln(suma:14:2,zi:4,'/',lu:2,'/',an:4);
                p:=p^.next;
                if wherey=25 then begin readkey;clrscr end
           end;
     repeat
           writeln('===> Depunere/Cheltuiala/Stop (D/C/S) ? ');
           c:=upcase(readkey)
     until c in ['D','C','S']
end;
procedure push;
begin
     new(p);p^.next:=vs;p^.suma:=d.suma;
     s:=s+d.suma;p^.data:=d.data;vs:=p
end;
procedure pop;
begin
     s:=s-d.suma;
     while vs^.suma<d.suma do begin
          d.suma:=d.suma-vs^.suma;p:=vs;
          vs:=vs^.next;dispose(p)
     end;
     vs^.suma:=vs^.suma-d.suma; vs^.data:=d.data
end;
begin
     vs:=nil; s:=0; consult;
     while c<>'S' do
          with d,data do begin
               repeat
                    write('Suma: ');readln(suma)
               until s-suma*byte(c='C')>=0;
               write('Ziua:');readln(zi);
               write('Luna:');readln(lu);
               write('Anul:');readln(an);
               if c='D' then push else pop;
               consult
          end
end.
```

Observație: în programul propus s-a considerat că datele depunerilor succesive sunt în ordine cronologică crescătoare; în caz contrar algoritmul implică o dizlocare de stivă cu inserarea depunerilor necronologice la locul lor.

Scanned with OKEN Scanner

10.9 (X)

Un fişier conţine pe mai multe perechi de linii numere reprezentând frecvenţele (pe o linie) şi duratele (pe următoarea) ale unor sunete ce formează câte un motiv muzical. Pe ultima linie a fişierului se află o suită de numere reprezentând o succesiune de motive muzicale, fiecare motiv fiind precizat prin numărul de linie pe care se găsesc notele. Să se genereze melodia corespunzătoare.

Exemplu: fişierul conţine liniile

```
330 294 262 330 350 392
 2   1   1   1   1   2
330 350 392 350 330 294 262
 1   1   1   1   1   1   2
330 262 262 262
 1   1   1   1
330 262 262
 1   1   2
0
2
1 2 1 2 5 3 4 2 3 4 2
```

```
motiv 1: mi re do mi fa sol

motiv 2: mi fa sol fa mi re do

motiv 3: mi do do do

motiv 4: mi do do

motiv 5: pauza

melodie pentru copii: Ceata lui Pitigoi
```

R 11.9

Datorită accesului secvenţial, fişierul dat nu poate fi folosit ca bază a motivelor melodice deoarece succesiunea lor nu coincide cu succesiunea liniilor din fişier, iar unele motive se vor şi repeta. Mai curând s-ar potrivi algoritmului de formare a melodiei o structură de tip fişier cu acces direct (cu tip), dar în acest caz se face multă risipă de memorie externă (spaţiu-disc) din cauză că motivele muzicale au lungimi diferite. Optăm deci pentru un ansamblu de liste ale căror adrese iniţiale se memorează într-un tablou de pointeri.

```
program motive_m;
uses crt;
const timp=200;
type adresa=^nota;
     nota=record
                n,d:word;
                urm:adresa
     end;
var f:text;nume_f:string;
    ini:array[1..10000]of adresa;
    nl,i:integer;
    p,q:adresa;
procedure eroare(a:string);
begin
     writeln(a);halt
end;
procedure motiv;
begin
     p:=ini[i];
     while p<>nil do begin
           sound(p^.n);delay(p^.d*timp);
           nosound;p:=p^.urm
     end
end;
```


Scanned with OKEN Scanner

```
procedure citeste;
begin
     if maxavail<=sizeof(nota) then
        eroare('Spatiu heap insuficient');
     new(q);read(f,q^.n);
     q^.urm:=nil;
end;
begin
     writeln('Numele fisierului (implicit cu extensia muz):');
     readln(nume_f);
     if pos('.',nume_f)=0 then nume_f:=nume_f+'.muz';
     assign(f,nume_f);reset(f);nl:=0;
     while not seekeof(f) do begin
           readln(f); inc(nl)
     end;
     if not odd(nl) then eroare('Fisier incomplet');
     reset(f);
     for i:=1 to nl div 2 do begin
         if seekeoln(f) then ini[i]:=nil
         else begin
                   citeste;ini[i]:=q;p:=q
              end;
         while not seekeoln(f) do begin
               citeste;
               p^.urm:=q;p:=q
         end;
         readln(f);p:=ini[i];
         while p^.urm<>nil do
               if seekeoln(f) then eroare('Durate insuficiente')
               else begin read(f,p^.d);p:=p^.urm end;
         readln(f,p^.d)
     end;
     while not seekeof(f) do begin
           read(f,i);
           if i<nl then motiv
     end
end.
```

11.10 (VII)

Să se realizeze o aplicaţie grafică şi apoi să se memoreze configuraţia unei zone dreptunghiulare de ecran într-o matrice pătrată formată din elemente între 0 şi *getmaxcolor*, un element m[*i*, *j*] egal cu *k* având semnificaţia: punctul din ecran cu coordonatele *i* şi *j* (relative la colţul din dreapta-sus al dreptunghiului) are culoarea *k*. Dreptunghiul de imagine „decupat" va fi apoi desenat pe ecranul grafic gol în colţul din dreapta sus. Coordonatele colţului dreptunghiului şi dimensiunile în pixeli ale zonei se citesc de la tastatură.

R 11.10

Decuparea se va face prin „măturarea" zonei menţionate şi testarea imaginii cu funcţia *getpixel*, rezultatul testării fiind depus în tabloul menţionat. Redesenarea lui se face procedând invers: se baleiază tabloul şi se desenează puncte de culoarea specificată cu procedura *putpixel*. Dimensiunile tabloului fiind foarte mari, nu

Scanned with OKEN Scanner

putem să-i afectăm suficient spaţiu în zona variabilelor statice (în segmentul de date), deci vom aloca dinamic spaţiul necesar acestuia ca un tablou de *n* pointeri la zone în care vom memora câte o linie:

```
program imagine;
uses graph;
const maxim=300;
type linie=array[1..2*maxim]of byte;
var x,y,l,h:word;
    i,j:word;
    dg,mg:integer;
    a:array[1..maxim]of ^linie;
procedure ia_imagine;
begin
     for i:=1 to h do begin
         new(a[i]);
         for j:=1 to l do a[i]^[j]:=getpixel(x+i-1,y+j-1)
     end
end;
procedure pune_imagine;
begin
     for i:=1 to h do
         for j:=1 to l do putpixel(i-1,j-1,a[i]^[j])
end;
procedure init;
var cale:string;
begin
     write('Calea driverului grafic (pentru C:\BP\BGI --> CR):');
     readln(cale);if cale='' then cale:='C:\BP\BGI';
     detectgraph(dg,mg);
     initgraph(dg,mg,cale);
     if graphresult<>grok then begin
        writeln('Cale incorecta');
     halt
     end;
end;
begin
     init;restorecrtmode;
     repeat
           write('Coord. coltului st.-sus al dreptunghiului:');
           readln(x,y)
     until (x<getmaxx)and(y<getmaxy);
     repeat
           write('Dimensiunile dreptunghiului (latime inaltime):');
           readln(l,h)
     until (x+l<=getmaxx)and(y+h<=getmaxy);
     setgraphmode(mg);
     for i:=0 to getmaxx div 20 do
         for j:=0 to getmaxy div 10 do begin
             setcolor((getcolor+1)*byte(getcolor<>getmaxcolor));
             line(getmaxx div 2,getmaxy div 2,i*20,j*10)
         end;
     ia_imagine;readln;cleardevice;
     pune_imagine;readln;closegraph
end.
```


Scanned with OKEN Scanner

Observaţii: micşoraţi valoarea constantei *maxim* şi declaraţi un tablou: `array[1..2*maxim,1..maxim]of byte` cu care urmaţi acelaşi algoritm sesizând diferenţele de timp pentru un dreptunghi de aceleaşi dimensiuni. Exersaţi apoi acelaşi efect folosind procedurile grafice *getimage* şi *putimage*:

```
program imagine1;
uses graph;
const maxim=300;
var x,y,l,h,i,j:word;
    dg,mg:integer;
    a:pointer;
procedure init;
......
begin
......
     getmem(a,l*h);
     getimage(x,y,x+l,x+h,a^);
     readln;
     cleardevice;
     putimage(0,0,a^,normalput);
     readln;
     closegraph
end.
```

11.11 (X)

Pentru un fişier cu tip ce conţine un număr nelimitat de nume de persoane (cu maxim 30 de caractere pentru fiecare nume) se creează un fişier index format numai din numerele de ordine ale înregistrărilor astfel încât, dacă s-ar scrie în ordinea numerelor din fişierul index s-ar obţine un şir ordonat alfabetic de nume de persoane. Să se redacteze programul care creează acest fişier şi care afişează numele indexate (aici cu semnificaţia „ordonate").

R 11.11

În condiţiile unui fişier cu un număr mic de persoane (care să se poată încărca toate în memorie) se poate recurge la o sortare prin numărare: pentru fiecare nume se numără câte nume sunt alfabetic mai mici şi adunând 1 se stabileşte indexul său.

Exemplificăm pentru şirul de numere diferite 33 8 2 11 92, în care 33 are indexul 4 deoarece există 3 numere mai mici decât el (8, 2 şi 11), 8 are indexul 2, etc., deci fişierul index conţine numerele 4, 2, 1, 3, 5.

Se ridică însă problema fişierelor mari al căror conţinut nu poate fi încărcat integral în memorie şi a căror prelucrare direct pe fişier implică n parcurgeri (ceea ce pentru un fişier cu $n = 5000$ de înregistrări deja înseamnă foarte mult). De aceea propunem o metodă ce combină sortarea prin inserţie cu sortarea prin interclasare şi care utilizează alocarea dinamică: se inserează într-o listă atâtea elemente câte încap în memorie împreună cu numerele lor de ordine originale şi se interclasează cu elementele unui fişier (iniţial vid), procedeul repetându-se până ce se epuizează fişierul de date. Apoi printr-o singură parcurgere se transferă indecşii (numerele lor de ordine) într-un alt fişier.

Scanned with OKEN Scanner

```
program indecsi;
uses crt;
type adresa=^pers;
     pers=record
                nume:string[30];
                nr:longint;
                urm:adresa
          end;
var f:text; i,ind:file of longint;
    g,g1:file of pers;
    p:pers; n:longint;
    q,r,cap:adresa;
procedure pune_in_lista;                              {insereaza la locul ei}
begin
     if (cap=nil)or(q^.nume<cap^.nume) then begin
             q^.urm:=cap;cap:=q
     end
     else begin
               r:=cap;
               while (r^.urm<>nil)and(q^.nume>r^.urm^.nume) do
                      r:=r^.urm;
               q^.urm:=r^.urm;r^.urm:=q
     end
end;
procedure scrie_din_fisier;
begin
     write(g1,p);read(g,p)
end;
procedure scrie_din_lista;
begin
     write(g1,q^);r:=q;q:=q^.urm;dispose(r)
end;
procedure interclaseaza;
begin
     rewrite(g1);seek(g,0);q:=cap;
     if not eof(g) then begin
        read(g,p);
        while (not eof(g))and(q<>nil) do
              if q^.nume<p.nume then scrie_din_lista
                                else scrie_din_fisier
     end;
     if eof(g) then while q<>nil do scrie_din_lista
     else begin
               repeat scrie_din_fisier until eof(g);
               write(g1,p)
     end;
     close(g);rewrite(g);seek(g1,0);
     while not eof(g1) do begin
           read(g1,p);write(g,p)
     end
end;
begin
     assign(f,'nume.txt');reset(f);
     assign(g,'nume.tip');assign(g1,'manev');rewrite(g);
     n:=0;clrscr;
```


Scanned with OKEN Scanner

```
      while not eof(f) do begin
            cap:=nil;
            while (not eof(f))and(maxavail>sizeof(p)) do begin
                  inc(n);new(q);readln(f,q^.nume);
                  q^.nr:=n;pune_in_lista
            end;
            interclaseaza
      end;
      close(f);close(g1);erase(g1);
      assign(ind,'nume.ndx');rewrite(ind);seek(g,0);
      while not eof(g) do begin
            read(g,p);write(ind,p.nr);
            writeln(p.nume:30,p.nr:10);
            if wherey=25 then begin
               write('Apasati o tasta');readkey;clrscr
            end
      end;
      close(g);erase(g);close(ind)
end.
```

10.12 (X – C, Concurs ACM, 1992)

Dându-se două caroiaje pătrate formate din 'X' şi '.' cu aceleaşi dimensiuni (maxim 10), se cere să se determine – dacă există – o transformare minimă prin care se ajunge de la caroiajul iniţial la cel final. Sunt permise următoarele operaţii:

– rotit 90 grade (caroiajul a fost rotit la dreapta 90 de grade)
– rotit 180 grade (caroiajul a fost rotit la dreapta 180 de grade)
– rotit 270 grade (caroiajul a fost rotit la dreapta 270 de grade)
– rotit vertical (caroiajul a fost reflectat în sus, în oglindă)
– regăsit (caroiajul final coincide cu cel iniţial)

R 10.12

Efectuarea succesivă a mai multor rotiri simple (*regasit* – funcţia identică sau *rotire_0, rotire_90* etc.) poate fi redusă la o singură rotire; în plus aplicarea transformării *vert* (rotire verticală) rezultatului altei transformări este totuna cu rotirea rezultatului unei transformări verticale, iar două rotiri verticale succesive aduc caroiajul în forma iniţială. Deci orice transformare minimă implică o rotire simplă sau o compunere *vert* cu rotire simplă. Propunem o rezolvare în care este „culeş" rezultatul unei funcţii de la adresa din heap la care se află, deoarece acest rezultat este un tablou şi limbajul PASCAL nu permite ca rezultat de funcţie decât un tip elementar, singura excepţie fiind tipul *string*. Puteţi realiza acelaşi program fără a lucra cu adrese, dacă veţi transforma funcţiile în proceduri, transferând tabloul rezultat prin lista de parametri (cu VAR), însă lizibilitatea programului principal scade.

Observaţii: rotit(..)^ ——> tabloul rezultat în urma aplicării funcţiei rotit;
rotit(.(rotit(..)^)^ ——> aplicarea funcţiei *rotit* rezultatului altei funcţii rotit.

```
program caroiaje;
type adresa_tab=^caroiaj;{adresa din memorie unde este un tablou}
     caroiaj=array[1..10]of string[22];
     rotire=(r90,r180,r270,vert);
     expresii=array[rotire]of string[9];
```


Scanned with OKEN Scanner

```
const celule=['.','X'];
      v:expresii=('90 grade','180 grade','270 grade','vertical');
var ini,fin:caroiaj;                              {Tablourile date}
    q:adresa_tab;  n,i:byte;
function regasit(a:caroiaj):boolean;
var i,j:byte;                          {Identificarea cu tabloul final}
begin
     regasit:=false;
     for i:=1 to n do if a[i]<>fin[i] then exit;
     regasit:=true
end;
function rotit(cod:rotire;a:caroiaj):adresa_tab;  {Rotire tablou}
var i,j:byte;x:adresa_tab;
begin
     new(x);
     for i:=1 to n do x^[i][0]:=chr(n);
     case cod of
          r90 :for i:=1 to n do
                   for j:=1 to n do x^[i,j]:=a[n-j+1,i];
          r180:for i:=1 to n do
                   for j:=1 to n do x^[i,j]:=a[n-i+1,n-j+1];
          r270:for i:=1 to n do
                   for j:=1 to n do x^[i,j]:=a[j,n-i+1];
          vert:for i:=1 to n do
                   for j:=1 to n do x^[i]:=a[n-i+1];
     end;
     rotit:=x
end;
procedure prelucrare;        {Aplicarea rotirilor neredundante}
var r:rotire;
begin
     if regasit(ini) then begin writeln('regasit.');exit end;
     for r:=r90 to vert do
         if regasit(rotit(r,ini)^)then begin
            writeln('rotit ',v[r],'.');exit
         end;
     for r:=r90 to r270 do
         if regasit(rotit(r,rotit(vert,ini)^)^)then begin
            writeln('rotit vertical si rotit ',v[r],'.');exit
         end;
     writeln('esec.')
end;
begin
     write('Latura caroiajelor=');readln(n);
     for i:=1 to n do begin
       write('Linia ',i,' a caroiajului initial:');readln(ini[i])
     end;writeln;
     { ............. idem pentru caroiajul final}
     write('Caroiajul initial a fost ');
     prelucrare;readln
end.
```

11.13 (X)

Se citesc vârfurile unui arbore binar şi descendenţii fiecărui vârf sub forma unor caractere (caracterul " " desemnând faptul că nu există un astfel de descendent). Să se afişeze parcurgerile în preordine, postordine şi inordine ale arborelui.

Scanned with OKEN Scanner

R 11.13

```
program parcurgeri;
type arbore=^varf;
     varf=record
                nod:char;
                st,dr:arbore;
          end;
var rad:arbore;
procedure citire(var e:arbore);
begin
     readln(e^.nod);e^.st:=nil;e^.dr:=nil;
     if e^.nod<>' ' then begin
        new(e^.st);new(e^.dr);
        write('Descend.stang pentru ',e^.nod);citire(e^.st);
        write('Descend.drept pentru ',e^.nod);citire(e^.dr)
     end
end;
procedure preordine(var e:arbore);
begin
     write(e^.nod,',');
     if e^.st^.nod<>' ' then preordine(e^.st);
     if e^.dr^.nod<>' ' then preordine(e^.dr)
end;
procedure postordine(var e:arbore);
begin
     if e^.st^.nod<>' ' then postordine(e^.st);
     if e^.dr^.nod<>' ' then postordine(e^.dr);
     write(e^.nod,',')
end;
procedure inordine(var e:arbore);
begin
     if e^.st^.nod<>' ' then inordine(e^.st);
     write(e^.nod,',');
     if e^.dr^.nod<>' ' then inordine(e^.dr)
end;
begin
    write('introduceti radacina ');new(rad);citire(rad);
    writeln('Preordine : '); preordine(rad);writeln(#8,' ');
    writeln('Postordine : ');ostordine(rad);writeln(#8,' ');
    writeln('Inordine : '); inordine(rad);writeln(#8,' ')
end.
```

Probleme propuse

11.14 (VIII)

Se citesc 3 numere şi se determină cel mai mic dintre ele folosind numai adresele lor din memorie.

11.15 (X)

Să se genereze o listă cu un număr aleator de componente, numere naturale, apoi să se afişeze numerele pare din listă.

Scanned with OKEN Scanner

11.16 (X)

Să se creeze o listă formată din numele, adresele şi telefoanele unor persoane. Se prelucrează lista permiţând alegerea dintr-un meniu a uneia dintre opţiunile : căutarea, adăugarea şi ştergerea datelor unei persoane, afişarea datelor tuturor persoanelor de pe o anumită stradă.

11.17 (X)

Se citesc de la tastatură n numere. Să se ordoneze folosind o listă şi să se afişeze în ordine crescătoare şi descrescătoare.

Indicaţie: Se va studia rezolvarea problemei 10.6.

11.18 (X)

Cărţile de joc între 2 şi 14 se aşează pe masă în cerc. Se dă numărul unei cărţi c şi numărul n. Se va elimina din cerc o carte începând de la cartea c numărând în continuare câte n cărţi. De la cartea eliminată se elimină a n-a carte etc. Jocul se continuă până ce rămâne o singură carte. Să se determine cartea rămasă.

Indicaţie: Se va folosi o listă circulară efectuând o poziţionare pe elementul cu eticheta c, apoi înaintări de câte n poziţii şi eliminarea elementului curent până ce eticheta elementului următor coincide cu eticheta elementului curent. Încercaţi şi o soluţie matematică de determinare a elementului eliminat fără simulare pe listă.

11.19 (IX – C)

Se consideră un teanc de n halate colorate, pentru fiecare halat, pornind de la baza teancului, dându-se culoarea sa, identificată printr-o literă mare. K funcţionari ($k \leq 12$), identificaţi prin numere de la 1 la k, iau la începutul programului din teanc, în această ordine, câte un halat, îl folsesc un timp, şi-l aşează la loc, deasupra teancului existent. Se precizează pentru fiecare funcţionar i durata d_i de utilizare a halatului. Dacă doi funcţionari aduc halatul în acelaşi timp, cel mai tânăr (cu număr de ordine mai mare) dă prioritate celui mai în vârstă.

a) Să se determine ordinea finală a halatelor în teanc.

b) Să se determine, dacă există, o altă ordine de „autoservire" a funcţionarilor astfel încât, la sfârşit, halatele să fie aşezate pe culori.

Exemplu: Pentru $n = 4$, halatele având culorile *A D A M* şi $k = 3$, duratele de utilizare fiind 4 9 2, se va afişa:

```
a) A M D A
b) 2 3 1 (-->A A D M)    sau    3 2 1 (-->A A M D)
```

11.20 (VII – C)

Se precizează operaţiile efectuate de un elev care rezolvă corect un exerciţiu de clasa I cu mai multe operaţii (adunări şi scăderi, fără paranteze). Fiecare operaţie dată are forma *termen* <*spaţiu*> *operaţie* <*spaţiu*> *termen*. Să se afişeze exerciţiul în întregime.

Indicaţie: Se construieşte un arbore binar strict. Se afişează parcurgerea arborelui în inordine.

Scanned with OKEN Scanner

BIBLIOGRAFIE

1. Atanasiu A., *Cum se scrie un algoritm? Simplu*, Editura Agni, Bucureşti, 1993.
2. Atanasiu A., *Probleme de Informatică*, Editura Petrion, Bucureşti, 1995.
3. Bălănescu T., Gavrilă S., Georgescu H., Gheorghe M., Sofonea L., Văduva I., *Programarea în limbajele Pascal şi Turbo Pascal*, Editura Tehnică, Bucureşti, 1992.
4. Dogaru O., Petcu D., Petrov Gh., *Turbo Pascal, Exerciţii şi probleme*, Editura de Vest, Timişoara, 1994.
5. Ivaşc C., Prună M., *Bazele informaticii (Grafuri şi elemente de combinatorică)*, Editura Petrion, Bucureşti, 1995.
6. Knuth D. E., *The Art of Computer Programming* (I), Addison-Wesley, Mass., 1968.
7. Livovschi Georgescu H., *Analiza şi sinteza algoritmilor*, Bucureşti, 1986.
8. Mateescu G. D., Mateescu I. C., *Analiză numerică.* Proiect de manual pentru clasa a XII-a, Editura Petrion, Bucurşti, 1995.
9. Mitrana V., *Provocarea algoritmilor*, Editura Agni, Bucureşti, 1994.
10. Niculescu R., Albeanu G., Domocos V., *Culegere de probleme Turbo Pascal*, Editura Tempus, Bucureşti, 1993.
11. Popovici C., State L., Georgescu H., *Bazele informaticii* – vol. I, Tipografia Univers, Bucureşti, 1990.
12. Rancea D., *Limbajul Turbo Pascal*, Editura Libris, Cluj, 1994.
13. Tomescu I., *Bazele informaticii.* Manual pentru clasele X–XI, Editura Didactică şi Pedagogică, Bucureşti, 1994.
14. Tomescu I., *Probleme de combinatorică şi teoria grafurilor*, Editura Didactică şi Pedagogică, Bucureşti, 1981.
15. Tudor S., *Manual de informatică pentru clasa a IX-a*, Editura Teora, Bucureşti, 1995.

* * * Gazeta de informatică, numerele 1, 7–8, 9, 10, 11 (1993), 1994, 1 (1996), Editura Libris, Cluj.

Scanned with OKEN Scanner

CUPRINS

Cuvânt înainte 3

I. Algoritmi elementari 5

II. Algoritmi cu tablouri 23

III. Şiruri de caractere şi mulţimi 43

IV. Structuri de tip înregistrare 63

V. Proceduri şi funcţii definite 78

VI. Unităţi de program 105

VII. Fişiere 128

VIII. Recursivitate 153

IX. Grafuri 168

X. Programare dinamică 196

XI. Alocare dinamică 219

Bibliografie 237

Scanned with OKEN Scanner

În aceeaşi colecţie au mai apărut manualele:

- Ion Diamandi – INFORMATICA ÎN 27 DE LECŢII (clasele V – VI)
- Marinel Şerban – QBASIC (gimnaziu)
- Cornelia Ivaşc, Mona Prună – BAZELE INFORMATICII (grafuri şi elemente de combinatorică – clasa a X-a)
- Cornelia Ivaşc, Mona Prună, Emanuela Mateescu – BAZELE INFORMATICII (grafuri şi elemente de combinatorică – caiet de laborator)
- Grigore Albeanu – SISTEME DE OPERARE (clasele X – XII)
- Marinel Şerban, Maria Ratyus – BAZELE INFORMATICII (WINDOWS, WORD pentru WINDOWS – clasa a XI-a)
- Mariana Panţiru – INFORMATICĂ ECONOMICĂ (clasele XI – XII)
- Radu Mârşanu, Adrian Vasilescu – INFORMATICĂ APLICATĂ (Alternativele A şi B – clasa a XII-a)
- Radu Mârşanu, Manole Velican – INFORMATICĂ APLICATĂ (Alternativa B – clasele XI – XII)
- George Mateescu, Ileana Mateescu – ANALIZĂ NUMERICĂ (clasa a XII-a)

ISBN 973-9116-05-1

Scanned with OKEN Scanner